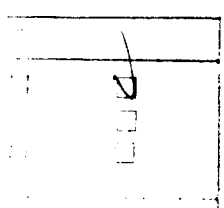
Perceptual Based Image Fusion

with Applications to Hyperspectral Image Data

THESIS
Terry Allen Wilson
Captain, USAF

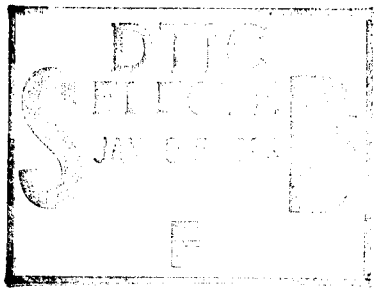
AFIT/GE/ENG/94D-32



DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GE/ENG/94D-32

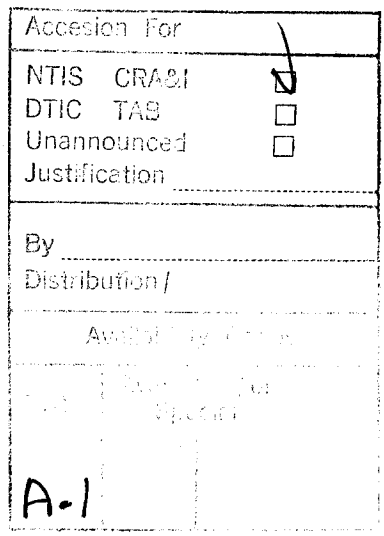


Perceptual Based Image Fusion

with Applications to Hyperspectral Image Data

THESIS
Terry Allen Wilson
Captain, USAF

AFIT/GE/ENG/94D-32



Approved for public release; distribution unlimited

AFIT/GE/ENG/94D-32

Perceptual Based Image Fusion with Applications to Hyperspectral Image Data

THESIS

Presented to the Faculty of the Graduate School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Electrical Engineering

Terry Allen Wilson, B.S.
Captain, USAF

December 1994

Acknowledgements

I would like to dedicate this work to my wife Deborah and my two sons Daniel and Tyler. As with all of my endeavors, their love and support sustains me. I would also like to thank my close friends Eric Baenen and Paul Barton and fellow students Lem Myers, Greg Alhquist, and Chris Eisenbis for their help and friendship. They helped to make my AFIT tour both tolerable and enjoyable.

Special thanks to John Colombi who provided valuable comments and suggestions that helped make this a better paper and Laura Suzuki who helped me understand some of the math early-on.

Finally, I would like to thank my advisor Dr. Steve Rogers. His motivation and guidance was a constant source of encouragement.

This research was supported by the Follow-On Tactical Reconnaissance System Program Office.

Terry Allen Wilson

Table of Contents

List of Figures

List of Tables

Abstract

Development of new imaging sensors has created a need for image processing techniques that can fuse images from different sensors or multiple images produced by the same sensor. The methods presented here focus on combining image data from the Airborne Visual and Infrared Imaging Spectrometer (AVIRIS) hyperspectral sensor into a single or smaller subset of images while maintaining the visual information necessary for human analysis.

Three hierarchical multi-resolution image fusion techniques are implemented and tested using the AVIRIS image data and test images that contain various levels of correlated or uncorrelated noise. Two of the algorithms are published fusion methods that combine images from multiple sensors. The third method was developed to fuse any co-registered image data. This new method uses the spatial frequency response (contrast sensitivity) of the human visual system to determine which parts of the input images contain the salient features that need to be preserved in the composite image(s).

After analyzing the signal-to-noise ratios and visual aesthetics of the fused images, contrast sensitivity based fusion is shown to provide excellent fusion results and, in every case, clearly outperformed the other two methods.

Finally, as an illustrative example of how the fusion techniques are independent of the hyperspectral application, they are applied to fusing multiple polarimetric images from a Synthetic Aperture Radar to enhance automated targeting techniques.

Perceptual Based Image Fusion
with Applications to Hyperspectral Image Data

I. Introduction

1.1 Introduction

Development of new imaging sensors has created a need for image processing techniques that can fuse images from different sensors or from multiple images produced by the same sensor [5, 2, 18, 19, 9]. For example, the Air Force is fusing information from multiple sensors to obtain a multi-spectral analysis of potential targets. The advantage of multi-spectral data is that it provides better target detection and identification than a single wide-band sensor [5, 14]. This allows a flexibility in choosing a particular narrow-spectral-band for individual types of targets [14]. The disadvantage with using multiple sensors to obtain a multi-spectral signature is that it is difficult and sometimes impossible to fully register the different input sources. Usually the different sensor recordings will differ in scale, rotation, or shift. To overcome the multi-sensor problems, research is being conducted with single sensors that simultaneously collect data in several bands. The Airborne Visual and Infrared Imaging Spectrometer (AVIRIS) is an example of a sensor that simultaneously records information in hundreds of spectral bands [14]. However, there is a price to pay for the fully registered hyperspectral data. Three particular problems, resulting from the use of the AVIRIS hyperspectral sensor, will be addressed in this thesis.

The first problem is what to do with all of the information generated by hyperspectral sensors. For example, a single AVIRIS sensor image requires approximately 140 megabytes of disk storage. However, only a fraction of that data provides unique or usable information about a target area. The rest is redundant information along multiple bands or is noise due to atmosphere. Thus, it would be beneficial to extract the relevant information from the

hyperspectral images. The need to interpret hyperspectral sensor data by photo analysts poses the second problem. Since a hyperspectral sensor records information along hundreds of spectral bands, each of these bands generates a single image for the analysts to interpret. Because human operators cannot physically or mentally integrate information from multiple source images [2, 18, 19], a method to fuse the relevant information into a single image, or at least a smaller subset of images, is needed.

A third problem results when an automated targeting system is used to find targets in remote sensor data. These automated systems are inundated with the vast amounts of information that modern imaging sensors produce about a target area, including many irrelevant details. This increases the chances for false alarms and missed detections. For example, the Air Force is investigating the idea of using different polarimetric orientations from a Synthetic Aperture Radar (SAR) for automated target recognition systems. Therefore, a fusion method that represents or preserves the details in the input images that are most relevant to the task at hand (i.e., target detection) and at the same time, provides better target detail [5], is needed.

1.2 Problem Statement

Can the data from a hyperspectral sensor, or from a SAR sensor, be combined into a smaller subset of images and still maintain the information from the input sources necessary for human or machine analysis?

1.3 Scope and Assumptions

The scope of this thesis is to investigate the image fusion algorithms developed by Peter Burt [5] and Alexander Toet [2, 18, 19], and then develop a new fusion method. The image fusion algorithms will be analyzed using test images with known image characteristics, image data from the AVIRIS hyperspectral sensor, and SAR sensor data. It is assumed that the images to be fused will either come from a single sensor that produces fully registered images, or from multiple sensors that have been pre-registered. It is also assumed the data produced from the sensors may be treated as image data and that contrast sensitivity, as related to human

visual perception, is the key for determining the salient features of an image [12, 13, 10, 16]. Furthermore, it is assumed that the key to multi-image fusion is to fuse based upon the salient features, (i.e., pattern primitives), and not at the pixel level alone [3, 4, 5, 2, 18, 19, 9].

The usefulness of the algorithms will be evaluated in three different ways. First the algorithms' visual aesthetic performances will be evaluated by fusing a set of test images with known image characteristics. Second the algorithms' usefulness in fusing SAR data will be analyzed by fusing SAR data and then measuring the effects on the automated target recognition of the scenes. Third, the algorithms' ability to fuse different spectral bands from the AVIRIS hyperspectral sensor will be evaluated.

1.4 Approach/Thesis Organization

Chapter one described data processing problems generated by using hyperspectral sensors and how image fusion may help solve them. Specifically, it discussed the need for data compression and how human photo analysts and machine vision systems may benefit from image fusion. Chapter two provides background information describing Burt's and Toet's algorithms along with a discussion on the AVIRIS sensor and contrast sensitivity. Chapter three details the algorithm developed in this thesis for image fusion. Chapter four describes the results of image fusion on the test images, the SAR image data (including the automated target recognition results) , and the AVIRIS hyperspectral image data. Conclusions and recommendations are discussed in chapter five.

II. Background

2.1 Introduction

This chapter will provide the background material necessary to understand the algorithms developed in chapter three. Specifically, it will detail the AVIRIS hyperspectral sensor data, a Gaussian decomposition with max contrast fusion algorithm by Alexander Toet [2, 18, 19], a wavelet decomposition with match and saliency fusion algorithm by Peter Burt [5], and contrast sensitivity as related to images. The algorithms discussed in this chapter are implemented using Matlab version 4.2 and are referenced in Appendices D and E.

2.2 AVIRIS Hyperspectral Data

This section describes the AVIRIS hyperspectral sensor and how images are produced from its data. It also describes the AVIRIS hyperspectral image data used in this thesis.

AVIRIS is an imaging spectrometer that simultaneously collects spectral information in the visual to infrared ranges. It records information in 224 spectral bands that range from $0.4\mu m$ to $2.5\mu m$ in approximately $10nm$ increments with a pixel resolution of roughly 20 meters at operational altitude. The AVIRIS sensor is a "whisk broom" type sensor that simultaneously collects spectral data in the 224 bands by sweeping a narrow band sensor back and forth as it is flown over a target area. Each sweep, from left to right or right to left, is called a line. Each line may contain up to 614 pixels of data. Typically the AVIRIS sensor collects 512 lines of image data in a single pass over an area. Figure 2.1 gives a pictorial representation of the hyperspectral AVIRIS sensor data collected from a 110 square kilometer area around Moffett Field, CA.

Creating the hyperspectral set of images is a two step process. First, a spectrometer measures the electro-magnetic energy reflected or emitted from a surface. The sampled energy may be in the visual or infrared spectral bands. The spectral range depends on the type of sensor employed. Since the spectral or luminance characteristic of a surface depends on the object's composition, targets comprised of different materials will have different reflected/emitted

intensity values for a given spectral band. The result is a set of intensity values that represent the target area in each band sampled. After the sensor collects the image data, the next step is to convert the data into a set of images.

Converting the data into an image is done by treating each sample as a piece of image information called a pixel. Each sample or pixel represents a single intensity value associated with a physical location in the target area. For example, each pixel value, in the AVIRIS data investigated in this thesis, covers a 20 square meter area of ground. The size of the area represented by each pixel is a function of the type of sensor, bandwidth, and altitude. The pixel values are used to form an image cube, where the X and Y axis of the cube represent the pixel location, and the Z axis represents the intensity of the sample in that spectral band(Figure 2.1).

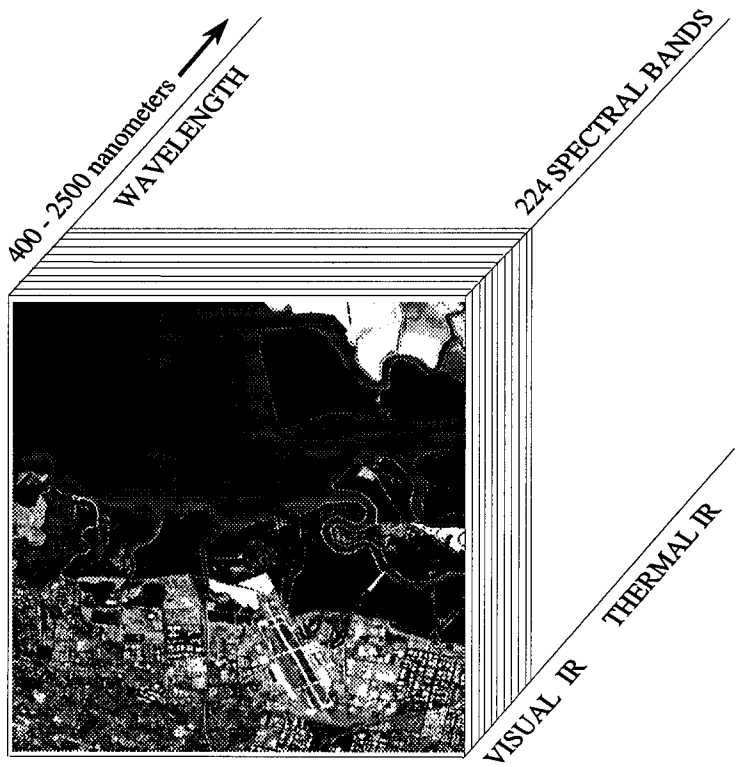


Figure 2.1 Hyperspectral Image Cube Representation taken from an AVIRIS Sensor, over Moffett Field, CA

The hyperspectral data used in this thesis is from an AVIRIS sensor that was flown over Moffett Field, CA. The sampled data representing the various intensities in each of the 224 spectral bands was used to create a hyperspectral image cube. The result is an image cube with 224 pictures of the same scene; one picture for each band. The image cube is like a deck of cards where each card represents a picture for a given band and the deck of cards is ordered by spectral band. The image, on the face of Figure 2.1, is an example of a picture from band 30. Band 30 represents the spectral information in the 0.67μm to 0.68μm bandpass range. It clearly shows the unique reflectance for the different types of materials: land, water, runway, etc.

2.3 Hierarchical Image Fusion

2.3.1 Gaussian Decomposition with Max Contrast Fusion. This section describe a multi-resolution image fusion technique developed by Alexander Toet [2, 18, 19] that uses maximum contrast as a criteria to preserve image details.

Toet's method involves a four step approach. The first step, which produces a multi-resolution pyramid, is depicted in Figure 2.2. The **reduce** function represents a convolution with a 5×5 filter and then a subsampling of the filtered image. First the input image is convolved with a 5×5 matrix (filter). A typical filtering window proposed by Toet [2, 18, 19] is represented by the following 5×5 weight matrix:

$$w = \begin{bmatrix} 0.04 & 0.10 & 0.08 & 0.10 & 0.04 \\ 0.10 & 0.25 & 0.20 & 0.25 & 0.10 \\ 0.08 & 0.20 & 0.16 & 0.20 & 0.08 \\ 0.10 & 0.25 & 0.20 & 0.25 & 0.10 \\ 0.04 & 0.10 & 0.08 & 0.10 & 0.04 \end{bmatrix}$$

Figure 2.3 is a gray-scale plot of the weight matrix w after the resolution was enhanced by interpolating with a cubic spline. Toet states that this is a Gaussian-like filter, but the numbers he recommended for the 5×5 filter do not make a Gaussian filter, see Figure 2.3. He references

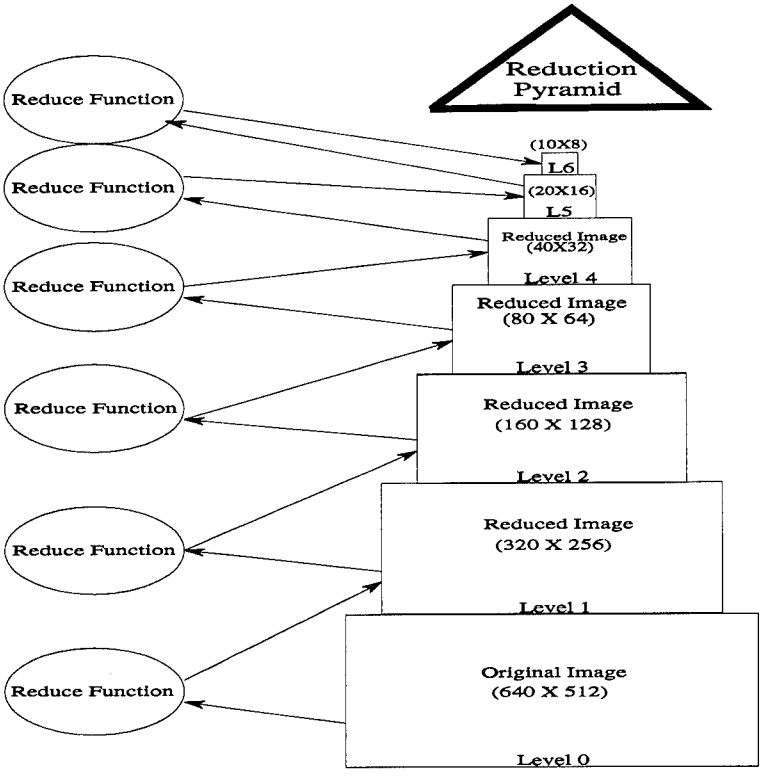


Figure 2.2 Multi-resolution decomposition algorithm that generates a reduction pyramid based on a single input image [2, 18, 19] .

the derivation for the filter from an article by Peter Burt ??. In that article, the filters produced are Gaussian-like, but the numbers recommended by Toet do not match the method presented in that article. The 5×5 filter presented by Toet is used in this thesis. After the image is filtered, it is down-sampled by a factor of two to produce an image that is half the resolution of the level below. Down-sampling by a factor of two is accomplished for two reasons. The first is the filtering operation removes the high frequency components so now the filtered image can be represented by only half the original pixel values. The second reason is that the same 5×5 filter can be used on the filtered and down-sampled image to further reduce the frequency components. Thus, the reduction pyramid contains a set of low-pass-filtered versions of the input image each with a band-limit one octave lower than the level below [2, 18, 19]. They are band-limited in one octave increments because the down-sampling cuts the frequency content by half each time.

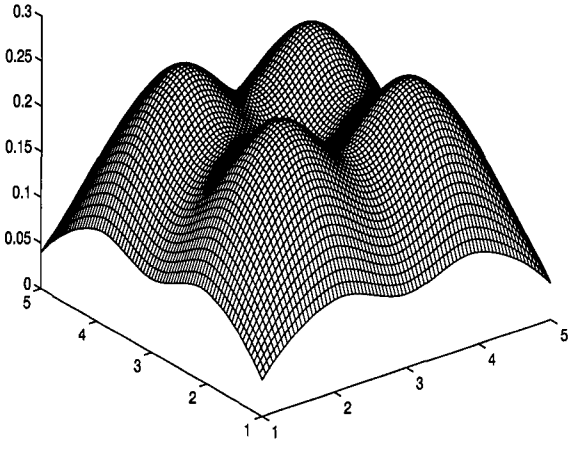


Figure 2.3 Resolution enhanced image representation of Alexander Toet's weight matrix w

The method of filtering and down-sampling is given by the equation,

$$P_l(i,j) = \sum_{m,n=-2}^{2} w(m,n) P_{l-1}(2i+m, 2j+n) \tag{2.1}$$

where P_l represents the (reduced) output level and P_{l-1} represents the (higher resolution) input level. The indices on the sum define the size of neighborhood that will be weighted to obtain the filtered image. Thus, the weight matrix w, defined above, is the convolution mask that is used to filter the image. Down-sampling, by a factor of 2, is accomplished by selecting every other point in the filtered image. By successively filtering and down-sampling, an image pyramid that has the original image at the pyramid base with successive levels that are filtered and down-sampled versions of the level below is generated. Figure 2.4 is a pictorial example of a three level pyramid. The first level is the original input image. The second is the filtered and down-sampled version of the input image. The third level is the filtered and down-sampled version of level two.

The second step in the process uses the reduction pyramid created in step one to create a ratio pyramid. Toet calls this a Ratio-of-Low-Pass (ROLP) [2, 18, 19]. The ratio pyramid is created by a point-by-point division of the lower level of the reduction pyramid by the expanded upper level of the same pyramid (Figure 2.5). The **expand** function is as follows:

$$P_{l,k}(i,j) = 4 \sum_{m,n=-2}^{2} w(m,n) P_{l,k-1}(\frac{i+m}{2}, \frac{j+n}{2}) \tag{2.2}$$

where $P_{l,k}$ represents the expanded output level, w is the same weight matrix as in Equation 1, $P_{l,k-1}$ represents the input level, and only integer indices contribute to the sum. The index k describes the number of **expand** operations that have been performed. In Toet's algorithm, there is only one **expand** operation per level.

The **expand** function is accomplished by padding every other column and row with zeros and then convolving the zero padded image with the weight matrix w. In effect, an upsampling by a factor of two is being accomplished. Now that the upper level has been upsampled, the two levels have the same number of pixel elements and a point-by-point division (of the gray-scale values) can be performed. The output of this stage is a ratio pyramid that represents the contrast information in the image at varying resolutions. The definition Toet gives for contrast [2, 18, 19] which is also defined in Peli's work on image

Reduced Image Level 2

Reduced Image Level 1

Original Image Level 0

Figure 2.4 Pictorial Representation of Alexander Toet's Multi-resolution Image Pyramid

analysis [12] is:

$$C = \frac{L - L_b}{L_b} = \frac{L}{L_b} - I$$

where C is the contrast, L is the local luminance in the image plane, L_b is the local background luminance, and $I(i,j) = 1$ for all i, j. Therefore, when C_i is defined as:

$$C_i = \frac{G_i}{EXPAND[G_{i+1}]} - I$$

the relationship between the ratio pyramid and the contrast is seen by:

$$R_i = C_i + I$$

.

This relationship is why Toet refers to the ratio pyramid as the contrast pyramid [2, 18, 19]. The number of levels in the ratio pyramid is one less than in the reduction pyramid, since each level (in the ratio pyramid) is created by dividing two levels (in the reduction pyramid). Also, the top level of the reduction pyramid and the ratio pyramid are the same, since there is no level above the top to divide by. Steps one and two are performed for every input image so that a ratio pyramid will be created for each image.

The third step in Toet's fusion method is to fuse all of the ratio pyramids into a single ratio pyramid which represents all of the input images. Figure 2.6 is a graphical depiction of the fusion stage. Toet uses the maximum contrast value to decide which value from the ratio pyramids will be retained. Since the ratio pyramid is a contrast representation [2, 18, 19, 12], a point-by-point comparison is made between all the ratio pyramids and the maximum value is retained for the composite ratio pyramid (Figure 2.6). The fourth and final step uses the fused ratio pyramid generated in step three to reconstruct the final fused image. Reconstruction is accomplished by reversing the steps it took to generate a ratio pyramid. The only difference between Figure 2.7, which shows the reconstruction, and Figure 2.5, which shows the generation of the ratio pyramid, is that instead of dividing by the expanded level

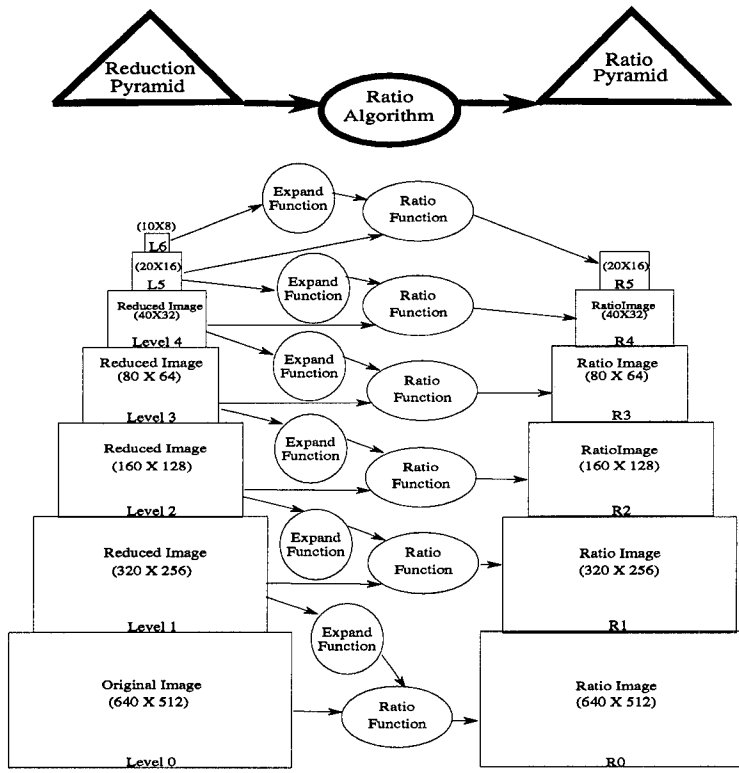


Figure 2.5 Multi-resolution contrast algorithm used to generate a ratio pyramid from an input reduction pyramid.

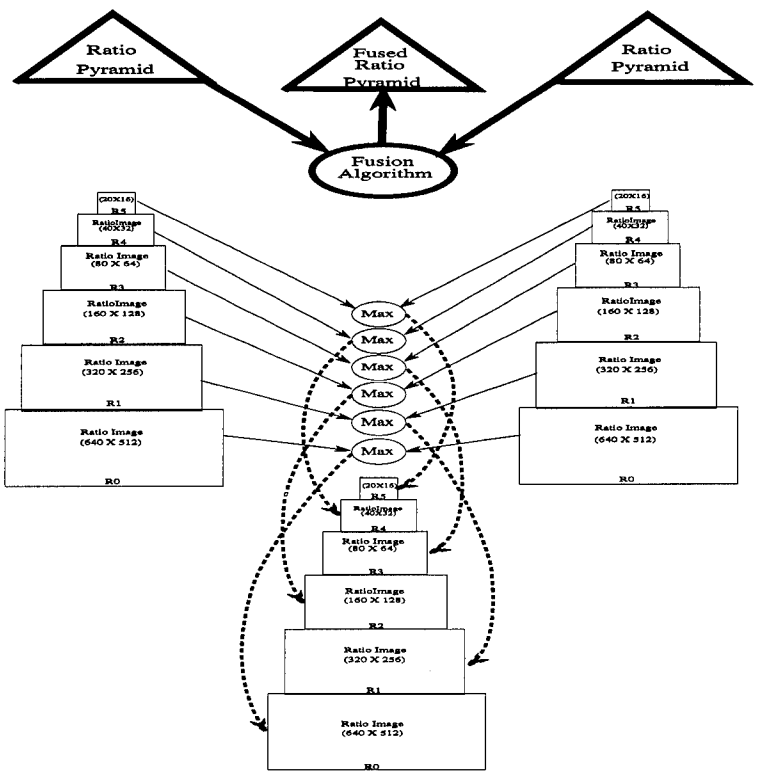


Figure 2.6 Multi-resolution contrast ratio fusion algorithm used to generate a fused ratio pyramid from a pair of ratio pyramids.

above you multiply. The end result is an image that is comprised of the maximum contrast details selected from the input images.

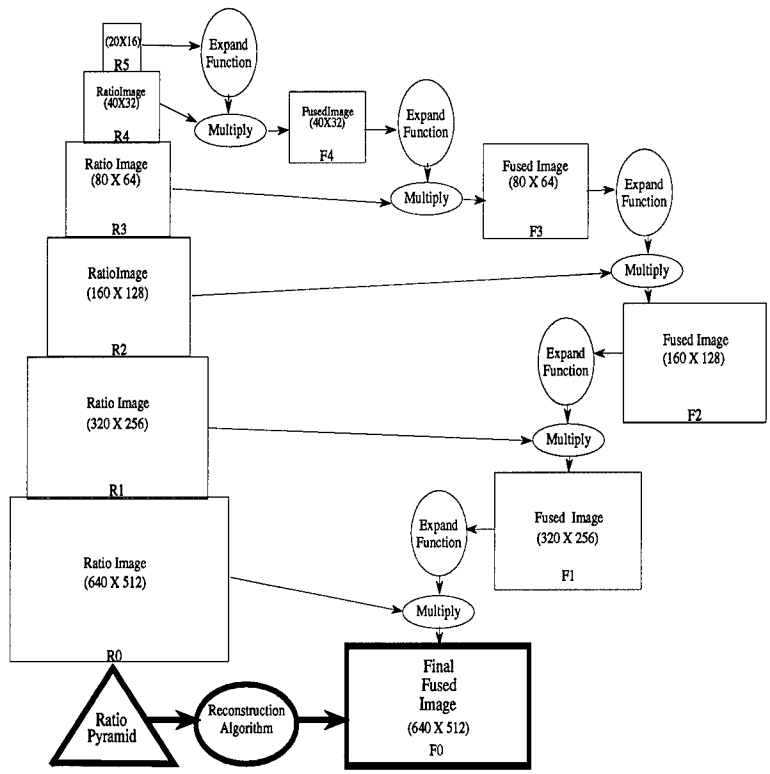


Figure 2.7 Multi-resolution reconstruction algorithm used to reconstruct the fused ratio pyramid into the final composite image.

2.3.2 Limits to Gaussian Decomposition with Max Contrast Fusion. Toet defends the decision to select the details for the composite image based upon maximum contrast by arguing that human vision is based upon contrast and that, by selecting details with maximum contrast, the resulting fused image will provide better details for the human analyst [2, 18, 19]. Although this is sound reasoning based upon the desire to present the human analyst with the best visual image, it does not account for the fact that a noisy image is typically of higher contrast than an image that is not. Therefore, Toet's method would select the noisier parts of the images to be retained in the composite, which presents a potential loss of information about desired targets. Figure 2.8, which is an example from chapter four, illustrates where

Toet's method would fail. The two images are from the same original image of a well known model named Lenna, but the second figure has added white Gaussian noise in one area of the image. Toet's method would select the area that has the added noise because it has higher contrast, even though the human analyst would select the less noisy parts of the two images. Thus, a method for selection that is based upon the perceptual sensitivity of the human visual system and not just pure contrast is needed.

Figure 2.8 Image of Lenna on the left and on the right is the same image with white Gaussian Noise added to one section.

2.3.3 Wavelet Decomposition with Match and Saliency Fusion. In the previous section an algorithm for image fusion by Alexander Toet was presented. Toet's work was based upon image fusion research published by Peter Burt in the early to middle 1980s. This section will discuss a recently published image fusion algorithm proposed by Peter Burt [5]. Burt's new method, which post-dates Toet's, is also a multi-resolution fusion algorithm that uses decomposition, fusion, and reconstruction.

First an image decomposition is performed. The first stage of Burt's image decomposition, which produces a multi-resolution image pyramid, is similar to Toet's reduction stage

above, see Figure 2.2. The **reduce** function filters an input image from a lower pyramid level with a 5×5 Gaussian filter. The filtered image is then down-sampled by a factor of two, by selecting every other pixel value, to produce an image that is half the resolution of the level below. The only difference between Burt's reduction method and Toet's is the type of 5×5 filter used to filter the successive levels of the image pyramid. A typical filtering window proposed by Burt [5], which is shown in a resolution enhanced gray-scale plot in Figure 2.9, is represented by the 5×5 weight matrix,

$$w = \dot{w} * \dot{w} = \frac{1}{256} \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

where,

$$\dot{w} = \frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

and $*$ is the convolution operator. Recalling the reduction method mentioned above for Toet [2], the effect of convolving each layer with the w weight matrix and then down-sampling by a factor of two is that a 2-D filtering operation is performed in the frequency domain.

The next stage in image decomposition is performed by extracting the orientation gradient details and a gross approximation from the multi-resolution pyramid. The detail extraction is not the same as what would be expected in a Mallat [11] decomposition, but the fact that the detail filters described below are compact, self-similar, and of many scales, they are considered wavelets [5]. The multi-scale orientation gradient details are extracted by using the filters, defined below, on multi-scaled versions of the image, Figure 2.10. The gross approximation is just the top level of the reduction (Gaussian) pyramid. Burt calls this step creating the orientation gradient pyramid [5]. It is called the orientation gradient because

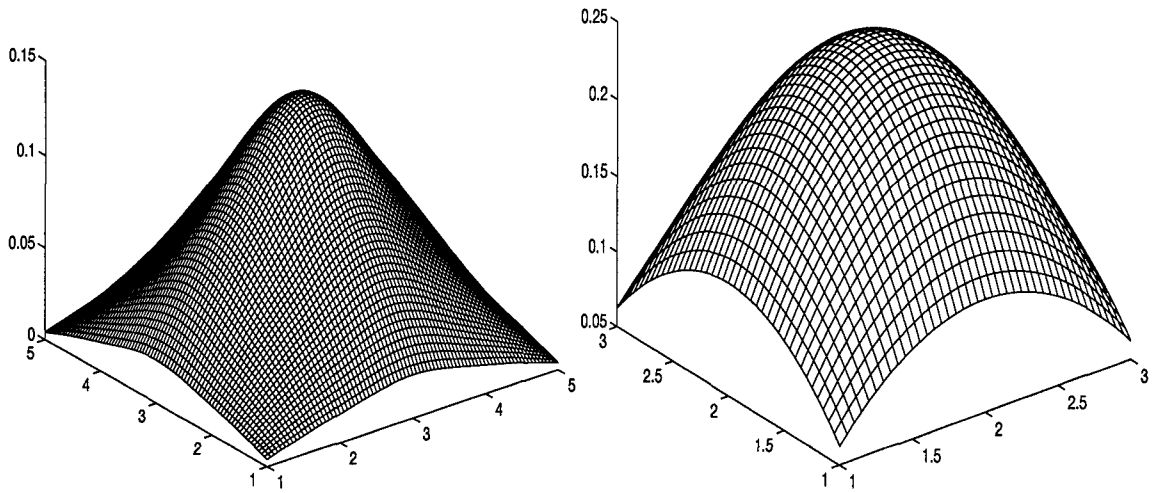


Figure 2.9 Resolution enhanced image representation of Peter Burt's weight matrix matrix w and $\dot{w}$

the basis functions used for detail extraction are gradients of Gaussian patterns [5]. They are called gradients of Gaussians because the gradient filters d_1 thru d_4, defined below, are used to extract information from the reduced (Gaussian) pyramid. The following equations define this stage of the decomposition:

$$D_{kl} = d_l * [G_k + \dot{w} * G_k]$$

$$d_1 = [1 \quad -1]$$

$$d_2 = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$$

$$d_3 = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

$$d_4 = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$$

where $*$ is the convolution operator, D_{kl} are the details for level k and orientation l, G_k is the level k input from the reduced image pyramid, and d_1 thru d_4 are the oriented gradient filters.

k is the level of the resolution from the Gaussian pyramid and the orientation l is simply the index of the d_1 thru d_4 filter used. Figure 2.11 presents the magnitude of the frequency response of filters d_1 thru d_4. Figure 2.12 is an example of one of the detail orientation gradient pyramids with three levels of resolutions and one orientation.

Figure 2.12 depicts one orientation from the oriented gradient detail pyramid that would be obtained from decomposing the image representing Band 30 of the AVIRIS sensor.

Now that the orientation details and gross approximation have been extracted, the second stage of Burt's fusion algorithm is ready to begin. This stage of Burt's algorithm is the main difference between Burt's method and Toet's [2, 18, 19]. Burt's fusion is performed in two parts. The first part is a comparison. If the details or "pattern primitives represented by the oriented gradients" [4, 5] from different pyramids (i.e. from different input images) are similar or match, they are averaged and the average value is retained. If they are very different, then the second part of the fusion method uses a saliency measure to decide which details will be retained. The actual equation for match is defined later, but in short, the weighted details from the orientation gradient pyramids are compared and a number is computed. The number ranges between a one and a minus one. A value of one means that the input gradient details are identical and a minus one means that they are identical but opposite in sign. If the match is above some threshold, say 0.85 [5], then the details from the input images for that particular location will be weighted as defined in the weighting equation below. If the match value is not above the threshold (the details are not similar enough), the detail that is most salient will be weighted with a value of one and the less salient detail will be weighted with a value of zero. The equation that defines **Saliency** for the image I is:

$$Saliency = S_I(m,n,k,l) = \sum_{\acute{m},\acute{n}} p(\acute{m},\acute{n})\, D_I(m+\acute{m}, n+\acute{n}, k, l)^2$$

where, m, n defines the location of the detail at a given level k in the oriented gradient pyramid, l defines the layer (i.e., which detail filter was used to extract the oriented detail) and $p(\acute{m},\acute{n})$ is a weight matrix with a neighborhood $(\acute{m},\acute{n})$ of 5×5, 3×3, or just the point itself. Thus,

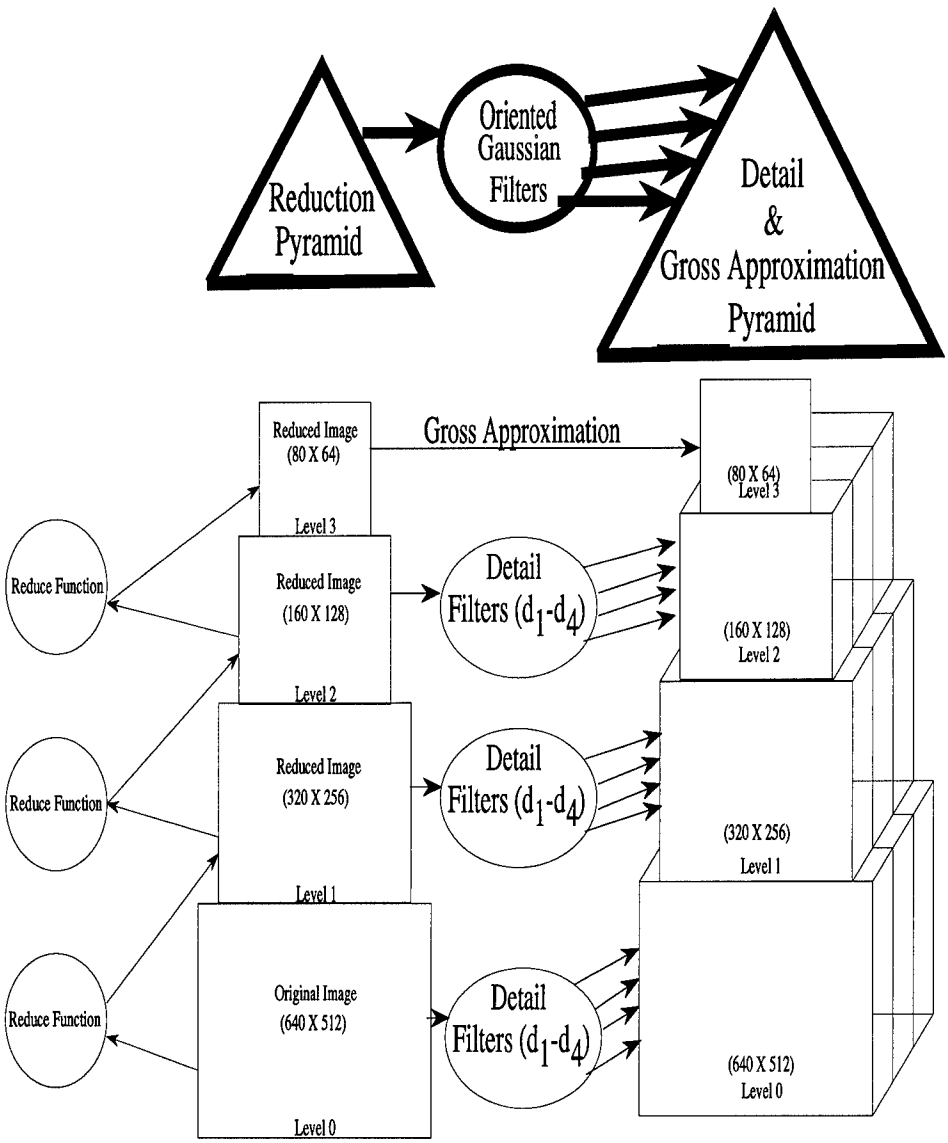


Figure 2.10 Orientation gradient pyramid algorithm used to extract the details and gross approximation from the reduction pyramid.

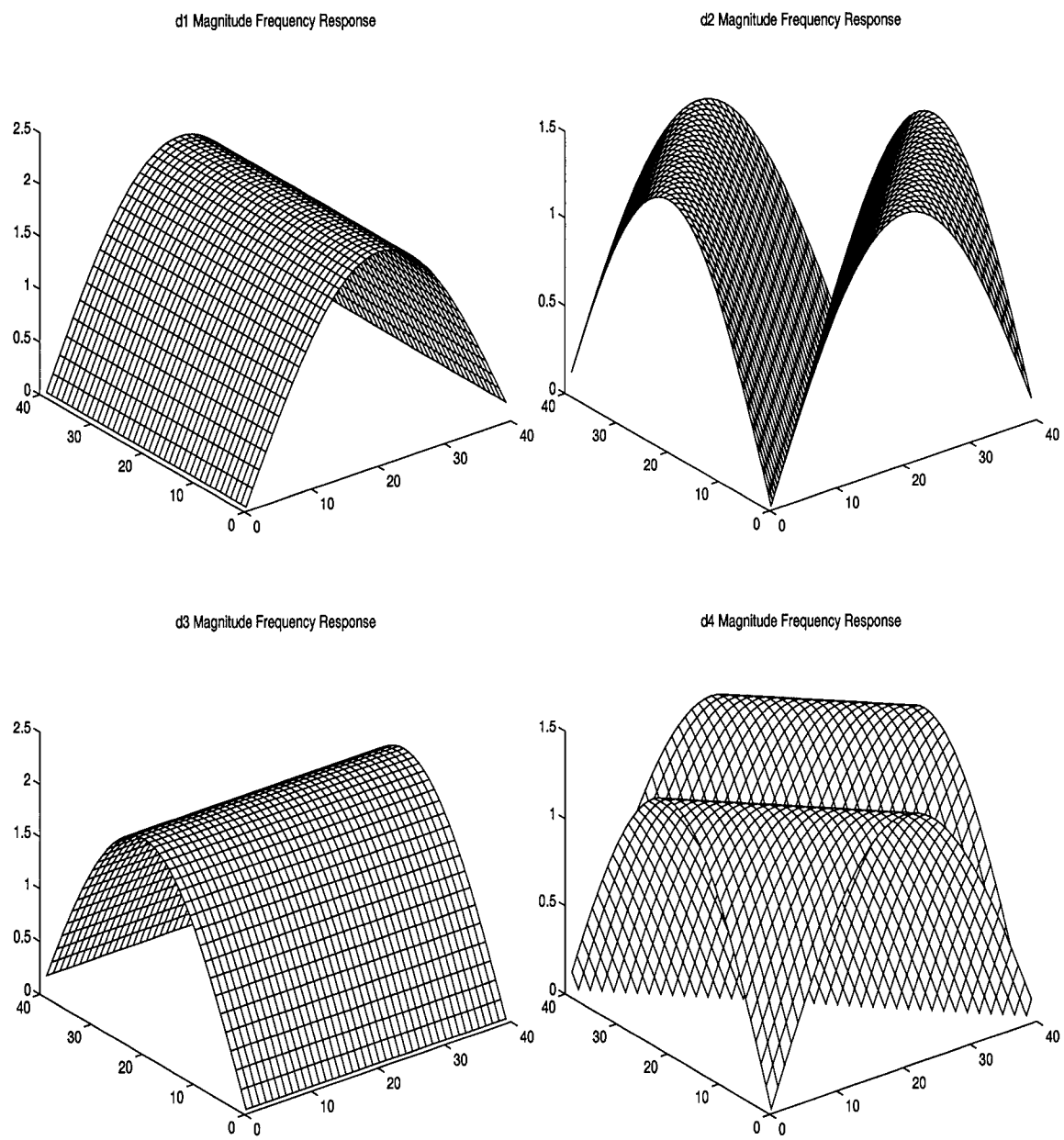


Figure 2.11 Filters d_1 Thru d_4 Magnitude Frequency Response.

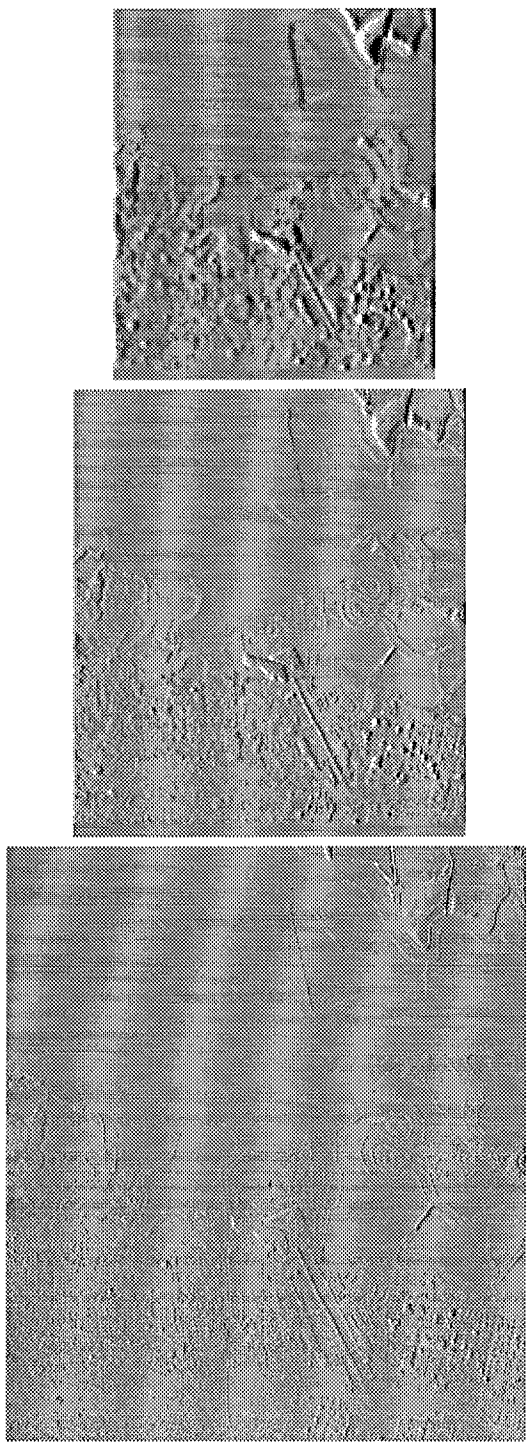


Figure 2.12 This figure provides an example of one particular orientation at three resolutions from the Orientation Gradient Pyramid which was obtained by using the d_1 (horizontal) gradient filter.

weights are chosen for the p matrix to determine how a particular neighborhood around each detail is weighted for importance. A typical weight matrix, which is recommended by Burt and used in this thesis, has all entries equal to one, so that saliency is the local gradient energy in some neighborhood, defined by the matrix p, around each point in the gradient pyramid. D_I represents the details from the orientation gradient pyramid from image I. The equation that defines **Match** between image A and image B is:

$$Match = M_{AB}(m,n,k,l) = \frac{2\sum_{\acute{m},\acute{n}} p(\acute{m},\acute{n}) D_A(m+\acute{m},n+\acute{n},k,l) D_B(m+\acute{m},n+\acute{n},k,l)}{S_A(m+\acute{m},n+\acute{n},k,l) S_B(m+\acute{m},n+\acute{n},k,l)}$$

Where D_A and D_B are the details from the orientation gradient pyramids from input images A and B. S_A and S_B are the salience values for the input images A and B. The match values range between the value 1 for identical input patterns and the value -1 for identical detail patterns that are opposite in sign.

In this way, images that may each contain a piece of the complete picture will contribute to the composite picture in some amount. The end result is an image that is composed of a combination of weighted averages of the details across several images.

The fused detail pyramid is created by summing the weighted details from image A and image B and choosing the gross approximation from one of the input sources, Figure 2.13. The weights and the weighted sum equations are defined as follows:

$$D_C(m,n,k,l) = w_A(m,n,k,l) D_A(m,n,k,l) + w_B(m,n,k,l) D_B(m,n,k,l)$$

$$w_{min} = \frac{1}{2} - \frac{1}{2}\left(\frac{1 - M_{AB}}{1-\alpha}\right)$$

$$w_{max} = 1 - w_{min}$$

Where D_C is the fused detail and w_A, w_B are the appropriate weight matrices. The weights are assigned based upon which input source had the largest salience value. If w_A had the larger salience then $w_A = w_{max}$ and $w_B = w_{min}$ otherwise, $w_B = w_{max}$ and $w_A = w_{min}$

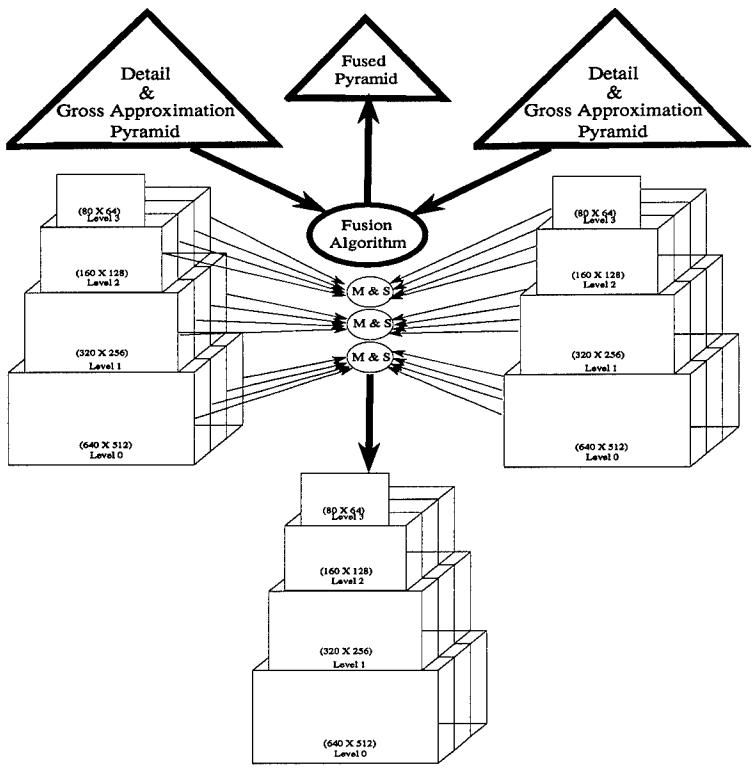


Figure 2.13 Burt fusion pyramid algorithm used to fuse images two at a time.

Once the oriented gradient pyramids have been fused into a single pyramid, the third and final stage of Burt's algorithm is performed. This is the reconstruction phase. Reconstruction is performed by combining the four layers of details into a single layer and then combining the gross approximation with the different levels of details, Figure 2.14. The first part of reconstruction results in a pyramid with multiple levels of combined details and a top level of a gross approximation, Figure 2.13. The final stage of reconstruction combines the gross approximation and the details to obtain the composite image, Figure 2.14. The method of reconstruction is further explained in chapter three.

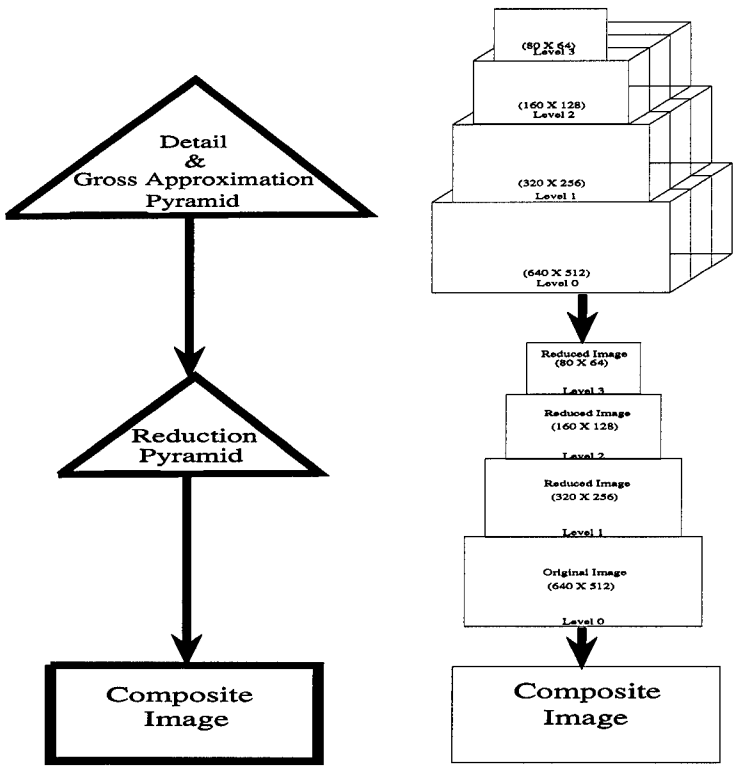


Figure 2.14 Burt reconstruction algorithm used to Combine the Gross Approximations and Details into a composite image.

2.3.4 Limits to Wavelet Decomposition with Match and Saliency Fusion. The fusion algorithm proposed by Burt has several advantages over the method proposed by Toet. Since it averages similar input sources, instead of just picking some maximum value, it offers

a potential for better noise reduction, which is shown in chapter four. It also allows the low contrast details to be preserved, if they are the salient features. The main disadvantage is that a template (weight matrix p) is needed to decide which features are salient. Since there are always problems with size, orientation, translation, etc., finding a template that will work well as a salient measure, will be very difficult if not impossible. One possible weight matrix, proposed by Burt, is a 3×3 matrix of ones; this allows the saliency to be based upon the local energy in the details. Although this provides a measure that has some useful applications, local energy in the details of some images may reflect a high value strictly due to noise in the image. Figure 2.15 is one such example. The two images are of the same scene, but the second image has some white Gaussian noise added to the center of the image. Burt's method would weight the noisy part of the second image more heavily than the corresponding non-noisy part of the first image, due to the energy in the noise. Another drawback to Burt's method is energy in the images that may not fall within human perception (see section 2.4) may play a large part in the decision of how the images will be fused. Again, see Figure 2.15. Therefore, a method that uses the strengths of Burt's fusion scheme, and still addresses the human visual system, is needed.

2.4 Contrast Sensitivity

Contrast is a measure of the difference in brightness across an image or scene. It is also the difference between the brightness of an object and its surroundings or background. For example, a black object on a white background would have a high contrast, whereas a gray object on a slightly lighter gray background would have a low contrast. Figure 2.16 represents two sinusoidally varying gratings one with a high contrast value of 0.8 and one with a low contrast value of 0.1, using the Michelson contrast equation defined later. As depicted in Figure 2.16 it is the relative differences in luminance between an object and its surroundings that determines the amount of contrast in a given scene. How well an individual can discern those relative differences in luminance determines the amount of contrast sensitivity an individual has. Contrast sensitivity is a measure of how a person

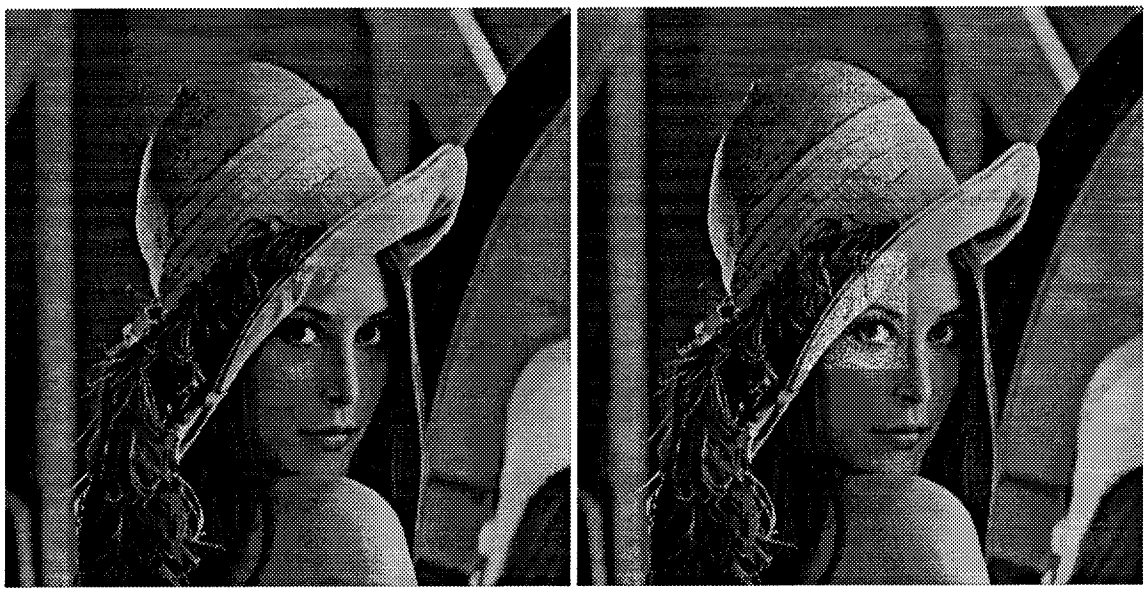


Figure 2.15 Image of Lenna on the left and on the left is the same image with white Gaussian Noise added to one section.

responds to contrast at threshold [17, 12]. Mathematically, contrast sensitivity is defined as the reciprocal of contrast.

Even though the definition for contrast seems fairly straight-forward, there are several methods to measure the contrast in a given scene [12]; each one has a unique application to a particular image analysis task. The measure of how the human visual system responds to contrast, i.e., contrast sensitivity, is a function of the spatial frequencies in the image [12, 17]. For example, it has been shown that the human visual system responds better, or is more contrast sensitive, to low spatial frequency components than it is to high spatial frequency components [17]. For instance, referring to the contrast sensitivity plot in Figure 2.19 [17], the sensitivity clearly peaks for the frequency ranges 2 through 10 cpd and then falls off sharply [17, 12].

When measuring visual acuity, the Snellen letters, black letters on a white background are used. Snellen letters are an example of high contrast, high frequency stimuli. Using this kind of test gives us no information about how well an individual can discern low contrast, low frequency stimuli such as a car on the road at night or a person in a dark movie

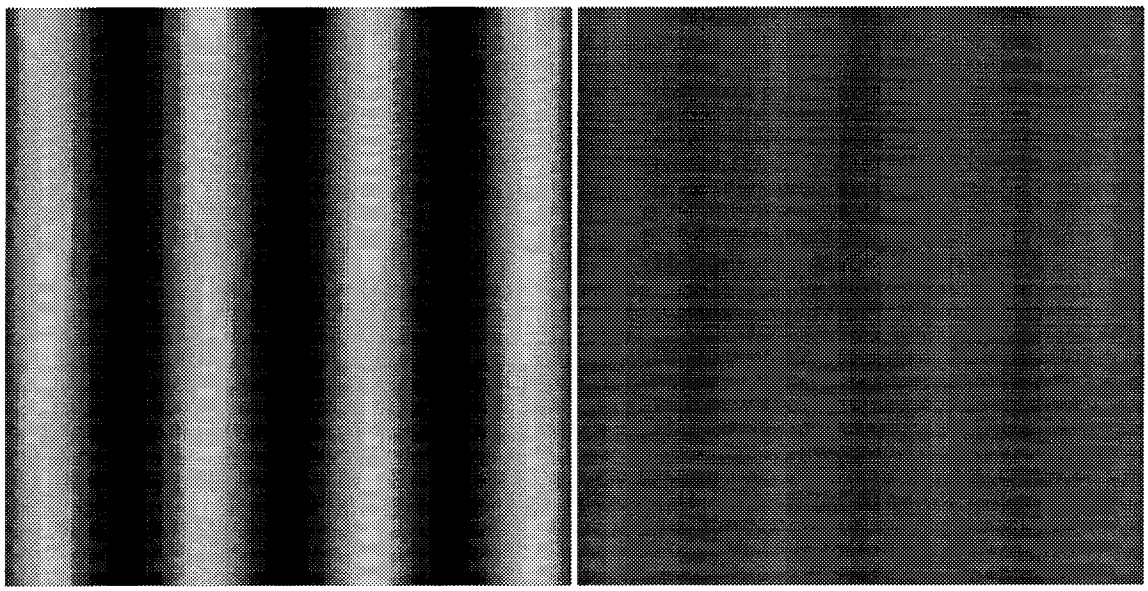


Figure 2.16 The figure on the left is an example of a low frequency high contrast sinusoidally varying gradient and the one on the right is a low frequency low contrast gradient

theater [17]. Examples of low contrast, low frequency stimuli include picking out a runway in the fog, looking for an individual in a dark theatre, or analyzing camouflaged targets in photo reconnaissance pictures. Because of the limited amount of information standard visual acuity tests provide, contrast sensitivity is becoming a more acceptable measurement criteria. Evans and Ginsburg showed that contrast sensitivity was better in determining a person's ability to discriminate highway signs and Stager and Hameluck showed that the same was true of air-to-ground search performance [17]. It is suspected that contrast sensitivity may even provide the Air Force with better information on an individual's visual capabilities than the standard visual acuity tests.

There are several methods to measure an individual's contrast sensitivity. One method is to present a subject with a series of bars at varying spatial frequencies and contrasts to determine, at each spatial frequency, what the necessary contrast is for the individual to see that there are bars in the image and not just a constant grey-scale image. This method measures an individual's contrast sensitivity threshold. Another method is to present a subject with a series of sinusoidally varying gratings, Figure 2.17, each with different spatial frequencies and

contrasts to measure the minimum contrast needed for an individual to detect the orientation of the grating. Figure 2.17 is an example of a high contrast vertically oriented test grating. To

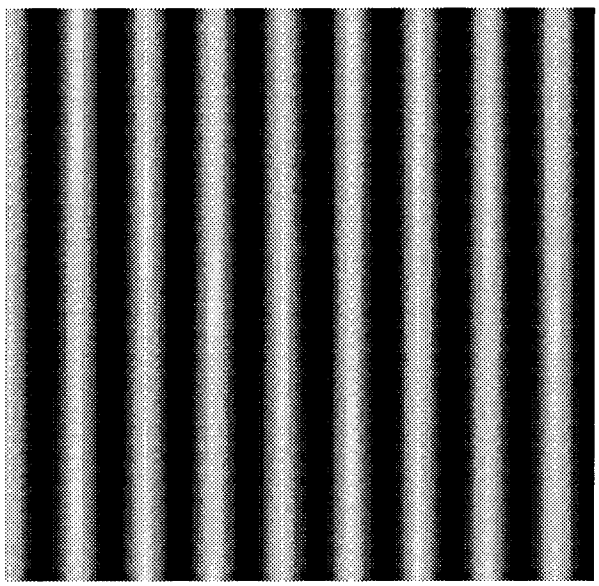


Figure 2.17 Example of a low frequency high contrast sinusoidally varying test gradient

determine the contrast for a particular grating the modulation contrast (or Michelson contrast) is generally computed [17, 12]. Michelson contrast is defined as:

$$Contrast = \frac{L_{max} - L_{min}}{L_{max} + L_{min}}$$

where L_{max}, L_{min} represent the maximum and minimum luminance values. For gray-scale images, L_{max}, L_{min} are the maximum and minimum values of the sinusoid. Spatial frequency is defined as the number of cycles per degree of visual angle. Therefore, assuming that a person is viewing an image on a computer screen 24 inches away, Figure 2.18, one degree of visual angle would relate to a physical viewing radius of:

$$Y = D \times \tan(\theta) = 24 \times \tan(1) = 0.2095 \in$$

where θ is the desired visual angle, and 24 is the distance in inches from the CRT. Therefore, on a typical CRT with a resolution of 1024×1280, the spatial frequency would be related to how many cycles are completed in roughly 40 pixels. This is why the window size of 40×40

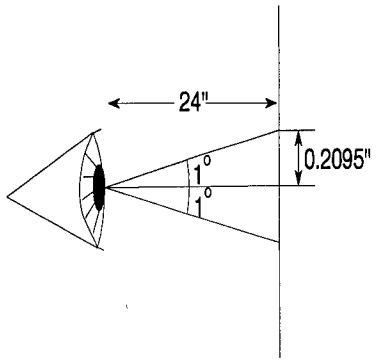


Figure 2.18 Diagram of how distance from CRT relates to Spatial Frequencies

is chosen for the image fusion algorithm developed in this thesis. The image fusion algorithm is designed to optimally fuse images for analysis on a CRT with a resolution of 1024×1280 and an average viewing distance of 24 inches from the screen. If a different presentation of the image is used, the window size can be altered to optimize for the viewing resolution and distance. The contrast sensitivity response used in this thesis is represented by the vector:

```
c = [0  50  100  150  160  150  145  142  164  130  120  110  100  90  80  70  66.5  63  59.5  56  52.5]
```

and is depicted in Figure 2.19. The *c* vector above represents the frequency response in cycles per degree of spatial resolution. Again, see Figure 2.18 for an understanding of how the distance from an object changes the spatial frequencies observed by the viewer. The contrast sensitivity response is provided as a one dimensional response to visual stimuli which vary only along one dimension (either vertically or horizontally) with no combination of vertical and horizontal. The image fusion algorithm developed in this thesis fuses images which are two dimensional. Thus, a weight matrix that provides the frequency response (i.e., contrast sensitivity) of the human visual system in a two dimensional response is needed. The one

dimensional c matrix is converted to the two dimensional response C by utilizing the following formula:

$$C(m,n) = sqrt\left[c(m)^2 + c(n)^2\right]$$

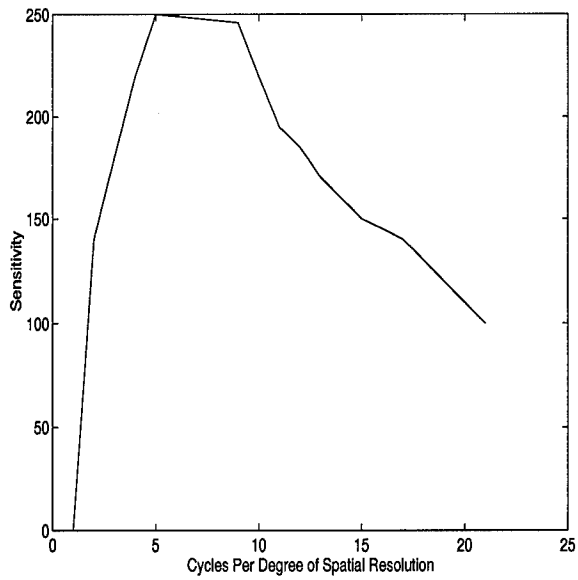


Figure 2.19 Contrast sensitivity response of the 95th percentile.

The success of this conversion relies on the assumption that the response of the human visual system is the same both horizontally and vertically [6, 8]. Also, the response can be extended to the other orientations by using a linear combination of the components of the horizontal and vertical responses [6, 8]. The two dimensional weight matrix C is shown below and is illustrated in the resolution enhanced Figure 2.20.

$$C = \begin{bmatrix}
0 & 140 & 180 & 220 & 250 & 249 & 248 & 247 & 246 & 220 & 195 & 185 & 170 & 160 & 150 & 145 & 140 & 130 & 120 & 110 & 100 \\
140 & 198 & 228 & 261 & 287 & 286 & 285 & 284 & 283 & 261 & 240 & 232 & 220 & 213 & 205 & 202 & 198 & 191 & 184 & 178 & 172 \\
180 & 228 & 255 & 284 & 308 & 307 & 306 & 306 & 305 & 284 & 265 & 258 & 248 & 241 & 234 & 231 & 228 & 222 & 216 & 211 & 206 \\
220 & 261 & 284 & 311 & 333 & 332 & 332 & 331 & 330 & 311 & 294 & 287 & 278 & 272 & 266 & 263 & 261 & 256 & 251 & 246 & 242 \\
250 & 287 & 308 & 333 & 354 & 353 & 352 & 351 & 351 & 333 & 317 & 311 & 302 & 297 & 292 & 289 & 287 & 282 & 277 & 273 & 269 \\
249 & 286 & 307 & 332 & 353 & 352 & 351 & 351 & 350 & 332 & 316 & 310 & 301 & 296 & 291 & 288 & 286 & 281 & 276 & 272 & 268 \\
248 & 285 & 306 & 332 & 352 & 351 & 351 & 350 & 349 & 332 & 315 & 309 & 301 & 295 & 290 & 287 & 285 & 280 & 276 & 271 & 267 \\
247 & 284 & 306 & 331 & 351 & 351 & 350 & 349 & 349 & 331 & 315 & 309 & 300 & 294 & 289 & 286 & 284 & 279 & 275 & 270 & 266 \\
246 & 283 & 305 & 330 & 351 & 350 & 349 & 349 & 348 & 330 & 314 & 308 & 299 & 293 & 288 & 286 & 283 & 278 & 274 & 269 & 266 \\
220 & 261 & 284 & 311 & 333 & 332 & 332 & 331 & 330 & 311 & 294 & 287 & 278 & 272 & 266 & 263 & 261 & 256 & 251 & 246 & 242 \\
195 & 240 & 265 & 294 & 317 & 316 & 315 & 315 & 314 & 294 & 276 & 269 & 259 & 252 & 246 & 243 & 240 & 234 & 229 & 224 & 219 \\
185 & 232 & 258 & 287 & 311 & 310 & 309 & 309 & 308 & 287 & 269 & 262 & 251 & 245 & 238 & 235 & 232 & 226 & 221 & 215 & 210 \\
170 & 220 & 248 & 278 & 302 & 301 & 301 & 300 & 299 & 278 & 259 & 251 & 240 & 233 & 227 & 223 & 220 & 214 & 208 & 202 & 197 \\
160 & 213 & 241 & 272 & 297 & 296 & 295 & 294 & 293 & 272 & 252 & 245 & 233 & 226 & 219 & 216 & 213 & 206 & 200 & 194 & 189 \\
150 & 205 & 234 & 266 & 292 & 291 & 290 & 289 & 288 & 266 & 246 & 238 & 227 & 219 & 212 & 209 & 205 & 198 & 192 & 186 & 180 \\
145 & 202 & 231 & 263 & 289 & 288 & 287 & 286 & 286 & 263 & 243 & 235 & 223 & 216 & 209 & 205 & 202 & 195 & 188 & 182 & 176 \\
140 & 198 & 228 & 261 & 287 & 286 & 285 & 284 & 283 & 261 & 240 & 232 & 220 & 213 & 205 & 202 & 198 & 191 & 184 & 178 & 172 \\
130 & 191 & 222 & 256 & 282 & 281 & 280 & 279 & 278 & 256 & 234 & 226 & 214 & 206 & 198 & 195 & 191 & 184 & 177 & 170 & 164 \\
120 & 184 & 216 & 251 & 277 & 276 & 276 & 275 & 274 & 251 & 229 & 221 & 208 & 200 & 192 & 188 & 184 & 177 & 170 & 163 & 156 \\
110 & 178 & 211 & 246 & 273 & 272 & 271 & 270 & 269 & 246 & 224 & 215 & 202 & 194 & 186 & 182 & 178 & 170 & 163 & 156 & 149 \\
100 & 172 & 206 & 242 & 269 & 268 & 267 & 266 & 266 & 242 & 219 & 210 & 197 & 189 & 180 & 176 & 172 & 164 & 156 & 149 & 141
\end{bmatrix}$$

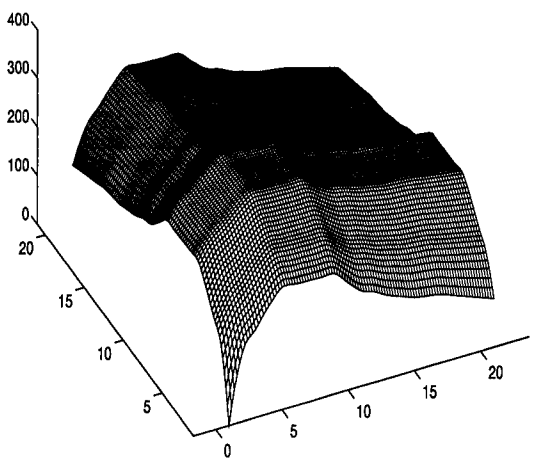


Figure 2.20 Contrast Sensitivity Response Weight Matrix C

Some research shows that an individual's perception of contrast will be different for contrasts that are above threshold [10, 13, 16] and the curve actually begins to flatten out as the contrast is increased. However, the fusion method proposed here is based upon the threshold

frequency response. Although it has not been tested, if the contrast sensitivity curve that matches the contrast in the images to be fused is available, replacing the contrast weighting matrix C with a new response curve, should only serve to further optimize the fusion method.

2.5 Conclusion

This chapter presented two image fusion techniques, one by Alexander Toet [2, 18, 19] and one by Peter Burt [5] and a discussion on contrast sensitivity. Contrast sensitivity is a measure of how the human visual system responds to certain visual stimuli. It is a valuable tool in analyzing the visual capabilities of a particular individual, and as such, will be invaluable when determining how images should be fused. The fusion methods presented had characteristics that make them both useful and practical. However, they each have limitations that need to be addressed. Toet's method, which is based upon maximum contrast, has the potential to lose information about low contrast targets. While Burt's method may overcome the contrast problem, it suffers from a need to find a good weight matrix p to define the saliency for a given application. The next chapter describes the fusion algorithm developed in this thesis and the test images that will be used to compare and contrast the different image fusion schemes.

III. Approach

3.1 Introduction

In the previous chapter, two multi-resolution image fusion algorithms were presented [18, 5], along with a discussion of how contrast sensitivity plays an important role in the human visual system. This chapter will describe the image fusion technique proposed in this thesis, the data used to evaluate the three image fusion algorithms, and the methods by which the three image fusion algorithms will be compared and contrasted. The contrast sensitivity algorithm discussed in this chapter is implemented using Matlab version 4.2 and is referenced in Appendix F.

3.2 Hierarchical Image Fusion with Contrast Sensitivity Saliency

This section describes an image fusion technique based upon previously successful multi-resolution decomposition and reconstruction methods [5, 15, 2, 18, 19, 9], with an added ability to tailor the selection criteria (i.e., what is salient between images) to the contrast sensitivity of the photo analyst. The basic approach, which is similar to the fusion methods discussed in chapter two, employs a multi-resolution algorithm that uses decomposition, fusion, and reconstruction. The decomposition and reconstruction phases will be identical to those employed by Peter Burt [5]. Those methods work well with an acceptable reconstruction error of approximately 2% of the original energy. The decomposition and reconstruction phases will be defined again here to provide a stand-alone picture of the fusion algorithm developed for this thesis.

First the image decomposition is performed. The first step in performing image decomposition is constructing the reduction or Gaussian pyramid, Figure 3.1.

The

$$5 \times 5$$

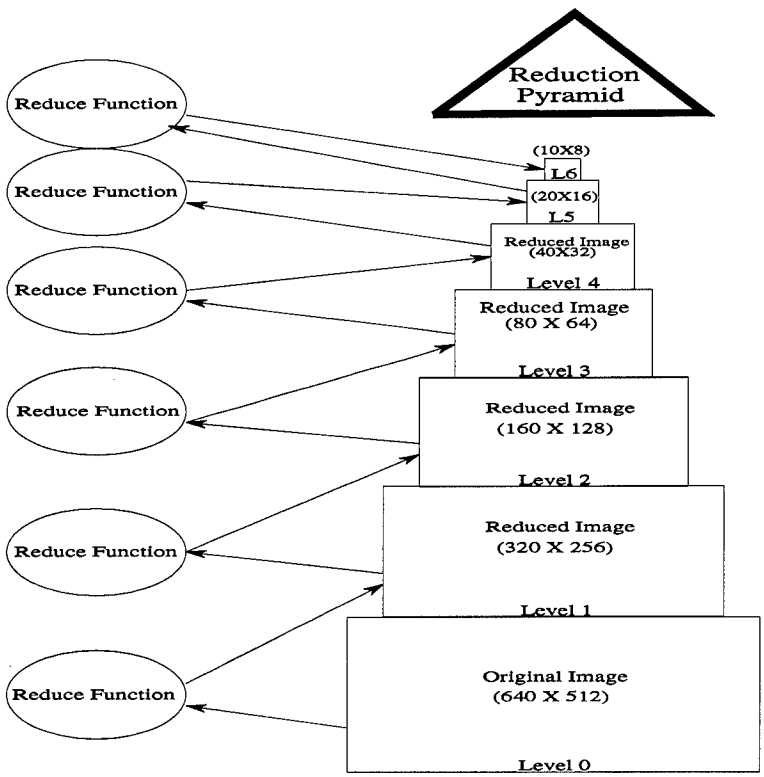


Figure 3.1 Multi-resolution decomposition algorithm that generates a Gaussian pyramid based on a single input image.

Gaussian kernel used in the reduction stage is represented by the weight matrix,

$$w = \dot{w} * \dot{w} = \frac{1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

where,

$$\dot{w} = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

The

$$5 \times 5$$

filter is resolution enhanced using a cubic spline to provide a better visual representation of how the filter looks, Figure 3.2. The method of filtering and down-sampling is defined by:

$$P_l(i,j) = \sum_{m,n=-2}^{2} w(m,n) P_{l-1}(2i+m, 2j+n) \tag{3.1}$$

where P_l represents the (reduced) output level and P_{l-1} represents the (higher resolution) input level, and the indices on the sum define the size of neighborhood that will be weighted to obtain the filtered image. Thus, the weight matrix w, defined above, is the convolution mask that is used to filter the image. Down-sampling, by a factor of 2, is accomplished by selecting every other point in the filtered image. By successively filtering and down-sampling, an image pyramid that has the original image as the pyramid base with successive levels that are low-pass filtered and down-sampled versions of the level below is generated. Recall, Figure 3.3 is a pictorial example of a three level pyramid. The first level is the original input image. The second level is the filtered and down-sampled version of level one (the input image). The third level is the filtered and down-sampled version of level two. The effect of

convolving each layer with the w weight matrix and then down-sampling by a factor of two is that a 2-D filtering operation is performed in the frequency domain.

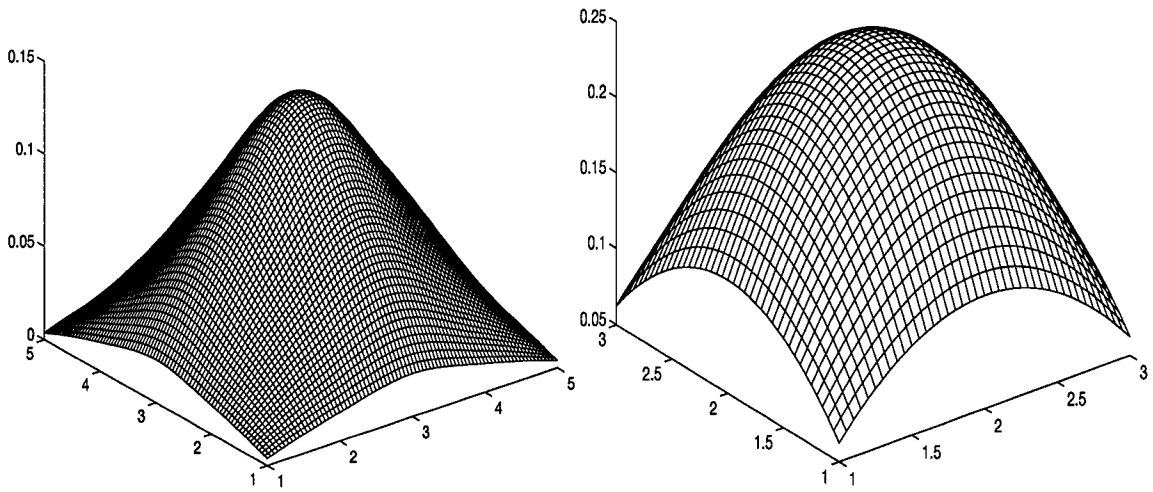
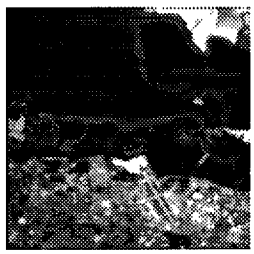


Figure 3.2 Image representation of resolution enhanced weight matrix matrix w and $\dot{w}$

The next stage in image decomposition is performed by extracting the orientation gradient details and a gross approximation from the multi-resolution pyramid. The detail extraction is not the same as what would be expected in a Mallat [11] decomposition, but the fact that the detail filters described below are compact, self-similar, and of many scales, they are considered wavelets [5]. The multi-scale orientation gradient details are extracted by using the filters, defined below, on multi-scaled versions of the image, Figure 3.4. The gross approximation is just the top level of the reduction (Gaussian) pyramid. Burt calls this step creating the orientation gradient pyramid [5]. It is called the orientation gradient because the basis functions used for detail extraction are gradients of Gaussian patterns. They are called gradients of Gaussians because the gradient filters d_1 thru d_4, defined below, are used to extract information from the reduced (Gaussian) pyramid. The following equations define this stage of the decomposition:

$$D_{kl} = d_l * [G_k + \dot{w} * G_k]$$



Reduced Image Level 2

Reduced Image Level 1

Original Image Level 0

Figure 3.3 Pictorial Representation of the Multi-resolution Image Pyramid

$$d_1 = \begin{bmatrix} 1 & -1 \end{bmatrix}$$

$$d_2 = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$$

$$d_3 = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

$$d_4 = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$$

where $*$ is the convolution operator, D_{kl} are the details for level k and orientation l, G_k is the level k input from the reduced image pyramid, and d_1 thru d_4 are the oriented gradient filters. The index k is the level of the resolution from the Gaussian pyramid and the orientation l is simply the index of the d_1 thru d_4 filter used. Figure 3.5 presents the magnitude of the frequency response of filters d_1 thru d_4. Figure 3.6 is an example of a single orientation from an orientation gradient pyramid with three different levels of resolution.

Figure 3.6 also depicts a single orientation from the oriented gradient detail pyramids that would be obtained from decomposing the image of Lenna.

Now that the oriented gradient pyramid has been formed for each input image, the next stage in the fusion process is performed. This is where the main difference lies between the method proposed here and Burt's method [5], which was presented in chapter 2. Burt uses a match and saliency measure, which is based upon the weighted energy in the detail domain, to decide how the oriented gradient pyramids will be combined. The method presented in this thesis uses an idea similar to Burt's match and saliency, but instead of using localized energy in the detail domain to compute the weighted averages, the method presented here uses a weighted energy in the perceptual domain; where the perceptual domain is based upon the frequency response (i.e., contrast sensitivity) of the human visual system.

The idea is that the human visual system responds to certain spatial frequencies differently than it does to others. In other words, some spatial frequencies are more important to the human observer, than others. Therefore, when evaluating whether two images would be perceived as different (determining a match value), and deciding which details are more

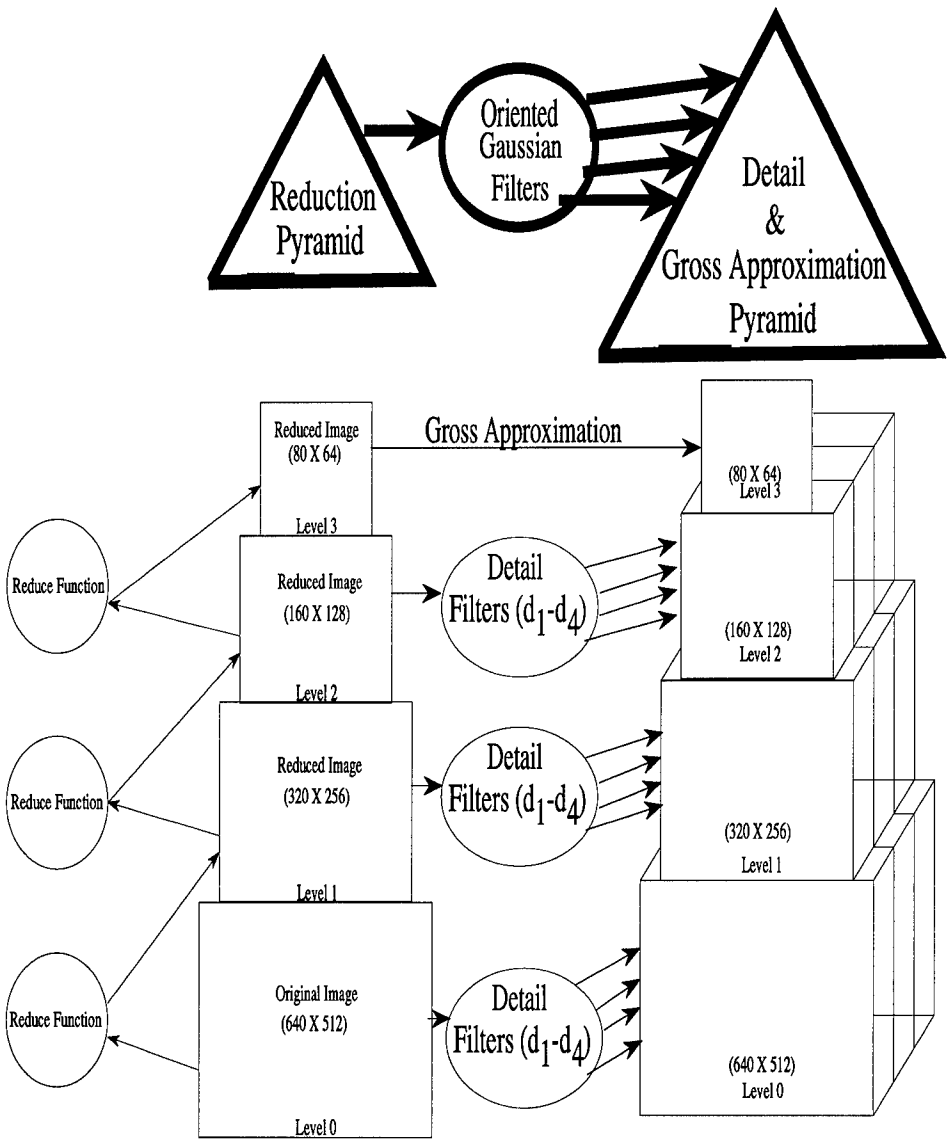


Figure 3.4 Orientation gradient pyramid algorithm used to extract the details and gross approximation from the reduction pyramid.

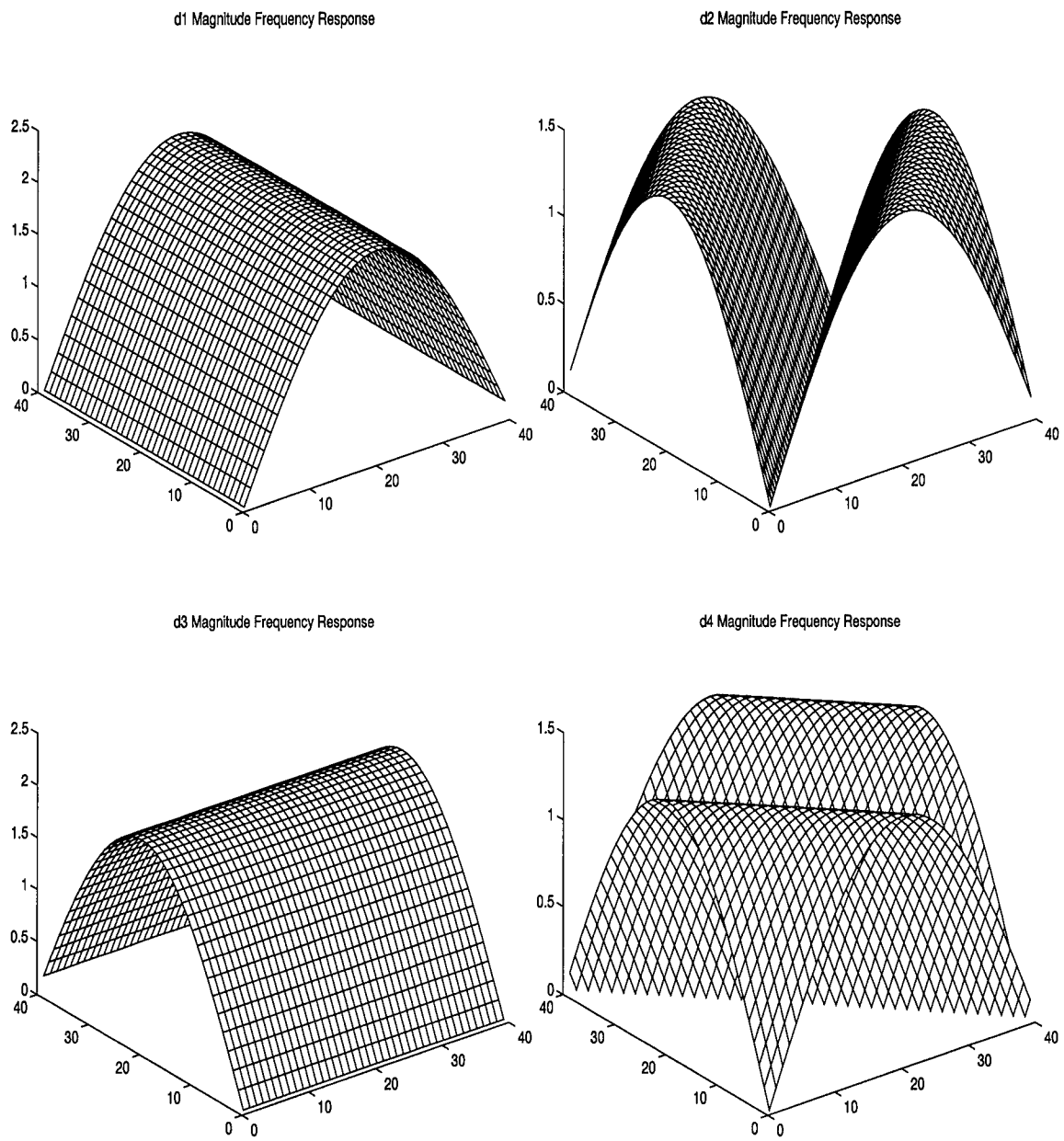


Figure 3.5 Plots of the Magnitude Frequency Response of filters d_1 thru d_4.

Figure 3.6 This figure provides an example of one particular orientation at three resolutions from the Orientation Gradient Pyramid which was obtained by using the d_1 (horizontal) gradient filter.

important (determining a saliency value), we need to base the evaluation on the perceptual abilities of the human observer. Thus, the criteria to decide which parts of the input images will averaged to form the composite image and which ones will not is determined by the contrast sensitivity response of the human analyst.

The fusion stage is performed in the following manner:

1. Corresponding neighborhoods from the orientation gradient pyramids to be fused will be compared using the contrast sensitivity response of the analyst.
2. If the corresponding neighborhoods differ by more than some threshold, the one that is more salient (i.e., has more energy in the contrast sensitivity frequency domain or "perceptual domain") will be retained in the fused orientation gradient pyramid.
3. If the perceived difference is less than some threshold, the two neighborhoods will be summed according to a weighted average.

First a relative perceptual distance D is computed for each layer of the orientation pyramid.

$$D = \frac{(S_A - S_B)}{(S_A + S_B)}$$

where D is the percent difference and S_A, S_B are the saliencies computed for a neighborhood from level k and orientation l of orientation gradient pyramids 1 and 2, respectively. Saliency is computed as the amount of perceptual energy in a given set of oriented details, as related to the frequency response C of the analyst. Saliency is defined by:

$$Saliency = \sum_{m,n} C\,(m,n)\, X_I(m\,n\,k,l)$$

where C is the contrast sensitivity weight matrix for a given analyst [17] and X_I is the magnitude of the energy normalized low frequency 2-dimensional Fourier components from some neighborhood at level k and orientation l of the orientation gradient pyramid. The indices m, n are defined by the window size.

A typical weight matrix C, representing the 2-dimensional frequency response for the human visual system, is depicted in Figure 3.7 and is defined as:

$$C = \begin{bmatrix}
0 & 140 & 180 & 220 & 250 & 249 & 248 & 247 & 246 & 220 & 195 & 185 & 170 & 160 & 150 & 145 & 140 & 130 & 120 & 110 & 100 \\
140 & 198 & 228 & 261 & 287 & 286 & 285 & 284 & 283 & 261 & 240 & 232 & 220 & 213 & 205 & 202 & 198 & 191 & 184 & 178 & 172 \\
180 & 228 & 255 & 284 & 308 & 307 & 306 & 306 & 305 & 284 & 265 & 258 & 248 & 241 & 234 & 231 & 228 & 222 & 216 & 211 & 206 \\
220 & 261 & 284 & 311 & 333 & 332 & 332 & 331 & 330 & 311 & 294 & 287 & 278 & 272 & 266 & 263 & 261 & 256 & 251 & 246 & 242 \\
250 & 287 & 308 & 333 & 354 & 353 & 352 & 351 & 351 & 333 & 317 & 311 & 302 & 297 & 292 & 289 & 287 & 282 & 277 & 273 & 269 \\
249 & 286 & 307 & 332 & 353 & 352 & 351 & 351 & 350 & 332 & 316 & 310 & 301 & 296 & 291 & 288 & 286 & 281 & 276 & 272 & 268 \\
248 & 285 & 306 & 332 & 352 & 351 & 351 & 350 & 349 & 332 & 315 & 309 & 301 & 295 & 290 & 287 & 285 & 280 & 276 & 271 & 267 \\
247 & 284 & 306 & 331 & 351 & 351 & 350 & 349 & 349 & 331 & 315 & 309 & 300 & 294 & 289 & 286 & 284 & 279 & 275 & 270 & 266 \\
246 & 283 & 305 & 330 & 351 & 350 & 349 & 349 & 348 & 330 & 314 & 308 & 299 & 293 & 288 & 286 & 283 & 278 & 274 & 269 & 266 \\
220 & 261 & 284 & 311 & 333 & 332 & 332 & 331 & 330 & 311 & 294 & 287 & 278 & 272 & 266 & 263 & 261 & 256 & 251 & 246 & 242 \\
195 & 240 & 265 & 294 & 317 & 316 & 315 & 315 & 314 & 294 & 276 & 269 & 259 & 252 & 246 & 243 & 240 & 234 & 229 & 224 & 219 \\
185 & 232 & 258 & 287 & 311 & 310 & 309 & 309 & 308 & 287 & 269 & 262 & 251 & 245 & 238 & 235 & 232 & 226 & 221 & 215 & 210 \\
170 & 220 & 248 & 278 & 302 & 301 & 301 & 300 & 299 & 278 & 259 & 251 & 240 & 233 & 227 & 223 & 220 & 214 & 208 & 202 & 197 \\
160 & 213 & 241 & 272 & 297 & 296 & 295 & 294 & 293 & 272 & 252 & 245 & 233 & 226 & 219 & 216 & 213 & 206 & 200 & 194 & 189 \\
150 & 205 & 234 & 266 & 292 & 291 & 290 & 289 & 288 & 266 & 246 & 238 & 227 & 219 & 212 & 209 & 205 & 198 & 192 & 186 & 180 \\
145 & 202 & 231 & 263 & 289 & 288 & 287 & 286 & 286 & 263 & 243 & 235 & 223 & 216 & 209 & 205 & 202 & 195 & 188 & 182 & 176 \\
140 & 198 & 228 & 261 & 287 & 286 & 285 & 284 & 283 & 261 & 240 & 232 & 220 & 213 & 205 & 202 & 198 & 191 & 184 & 178 & 172 \\
130 & 191 & 222 & 256 & 282 & 281 & 280 & 279 & 278 & 256 & 234 & 226 & 214 & 206 & 198 & 195 & 191 & 184 & 177 & 170 & 164 \\
120 & 184 & 216 & 251 & 277 & 276 & 276 & 275 & 274 & 251 & 229 & 221 & 208 & 200 & 192 & 188 & 184 & 177 & 170 & 163 & 156 \\
110 & 178 & 211 & 246 & 273 & 272 & 271 & 270 & 269 & 246 & 224 & 215 & 202 & 194 & 186 & 182 & 178 & 170 & 163 & 156 & 149 \\
100 & 172 & 206 & 242 & 269 & 268 & 267 & 266 & 266 & 242 & 219 & 210 & 197 & 189 & 180 & 176 & 172 & 164 & 156 & 149 & 141
\end{bmatrix}$$

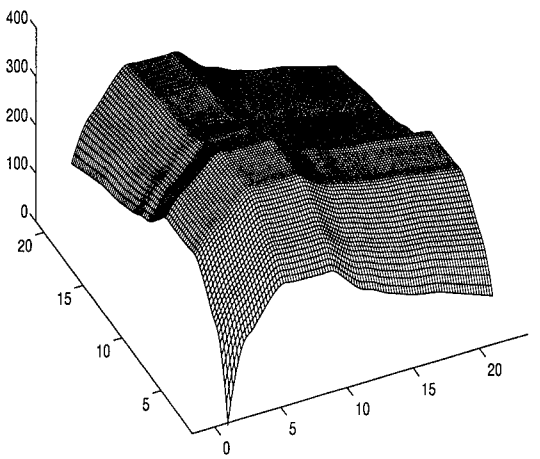


Figure 3.7 Median Contrast Sensitivity Response Weight Matrix C

The previous definition for the C matrix applies to the level zero version of the contrast sensitivity matrix. When the saliency is computed for successive levels of the oriented gradient

pyramid, the C matrix needs to be scaled appropriately. The level of scaling would correspond to the type of resolution scaling being performed on the original image. Therefore, if the image has been reduced three times, i.e., the fourth level of the oriented gradient pyramid, then the C matrix needs to be reduced three times as well. This allows the response of the contrast sensitivity function to remain at the same level of analysis as it did at the original level zero of the pyramid. Thus, to obtain the appropriate C matrix for a given level the relation:

$$C_k = Reduce^k(C)$$

where C_k is the C matrix used in the saliency formula for a given level k. The subscript k on the $Reduce_k$ function defines how many times the reduce function needs to be performed on the original C matrix.

The method for extracting the saliency is to first energy normalize each level and orientation of the oriented gradient pyramid by dividing by the square root of the sum of the squares at each level and orientation. Next, extract the magnitude of the Fourier coefficients of an m, n sliding window over each input level and orientation. The size of the window and the step size may vary. A 40×40 window and a step size of eight was used in this thesis. To extract the Fourier coefficients, the 40×40 window is passed over the image, according to the step size, and the Fourier transform is computed for each resulting 40×40 set of values. Then, the non-zero frequency components are energy normalized by dividing each coefficient by the square root of the sum of the squares of the non-zero frequency coefficients. The zero frequency component values are not used. This type of normalization allows images to be compared in the same manner as a photo analyst would. That is, the photo analyst's visual system automatically discounts the illuminant in images that have large differences in average intensity, but are of the same scene. They are perceived as equivalent. Figure 3.8 is an example of two images that are identical except for some average intensity offset.

After the percent difference D is computed, it is compared to some threshold T. If the difference is greater than T, the input detail with the higher saliency value will be retained for

Figure 3.8 Images only differ by some dc bias

the fused oriented gradient pyramid and the other detail will not. If the perceived difference is not greater than the threshold T, the input details will receive the following weights:

$$W_A = \left(1 - \frac{Saliency_B}{Saliency_A} * 0.5\right)$$

$$W_B = 1 - Weighting_A$$

The fused detail pyramid is created by summing the weighted details from image A and image B and choosing the gross approximation from one of the input sources, Figure 3.9. The weighted sum is defined as follows:

$$D_C = W_A D_A + W_B D_B$$

Here D_C is the fused detail and W_A and W_B are the appropriate weight matrices and D_A and D_B are the details from the oriented gradient pyramids from image A and B. The weighting is performed by a point-by-point multiplication of the weight matrix and the detail matrix.

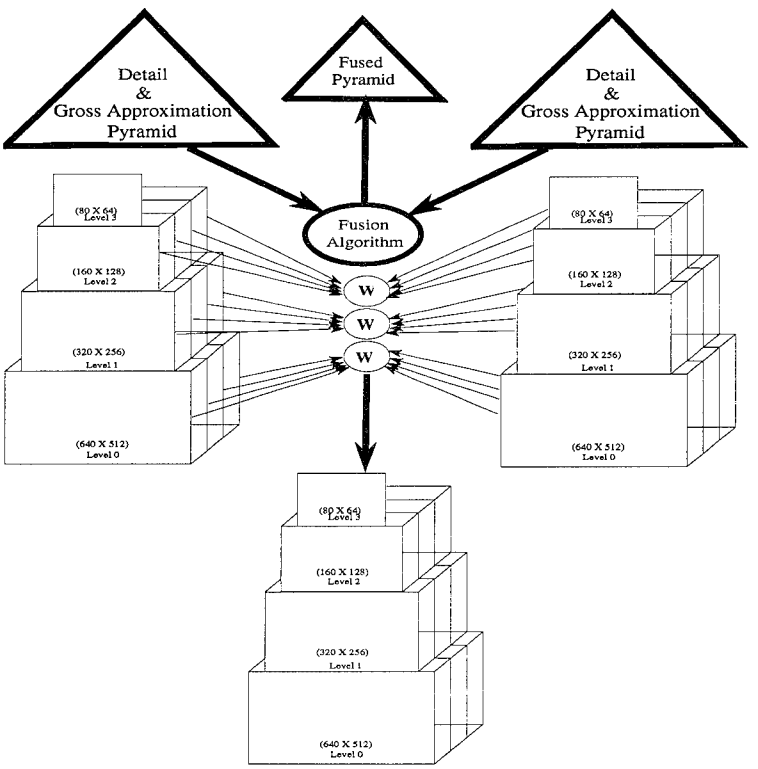


Figure 3.9 Fusion pyramid algorithm used to fuse images two at a time.

Once the gross approximation and detail pyramids have been fused into a single pyramid, the third and final stage is performed. This is the reconstruction phase. Reconstruction is performed by combining the four layers of details into a single layer and then combining the gross approximation with the different levels of details, Figure 3.10, to form the composite image. The order of reconstruction is

1. Convert the oriented gradient pyramid into the second derivative pyramid, or what Burt calls the oriented Laplacian pyramid [5].
2. Convert the oriented Laplacian pyramid into the FSD (filter-subtract-decimate) Laplacian pyramid [3].
3. Convert the FSD Laplacian pyramid into the RE (reduce-expand) Laplacian pyramid.
4. Convert the RE Laplacian pyramid into the Gaussian pyramid.
5. Recover the composite image from the Gaussian.

Converting the oriented gradient pyramid into the oriented Laplacian pyramid is represented by the following formula:

$$\vec{L}_{kl} = -\frac{1}{8} d_l * D_{kl}$$

where $\vec{L}_{kl}$ is the level k and orientation l of the oriented Laplacian pyramid, d_l is one of the detail filters described above for d_1 thru d_4, and D_{kl} is the input level k and orientation l from the oriented gradient pyramid.

Converting the oriented Laplacian pyramid into the FSD Laplacian pyramid is represented by the following formula:

$$L_k = \sum_{l=1}^{4} \vec{L}_{kl}$$

where L_k is the level k of the FSD laplacian pyramid and $\vec{L}_{kl}$ is the level k and orientation l of the oriented Laplacian.

Converting the FSD Laplacian pyramid into the RE (reduce-expand) Laplacian pyramid is defined as follows:

$$\tilde{L}_k \approx [1 + w] * L_k$$

Where $\tilde{L}_k$ is the level k of the RE (reduce-expand) Laplacian pyramid, L_k is the level k of the FSD laplacian pyramid, w is the 5×5 filter defined above, and δ is a matrix defined as follows:

$$\delta = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.2}$$

Converting the RE Laplacian pyramid into the Gaussian pyramid and then recovering the composite image from the Gaussian, is performed by repeatedly applying the following formula:

$$\hat{G}_k = \tilde{L}_k + 4w * [G_{k+1}]_{\uparrow 2}$$

Where $\hat{G}_k$ is the level of reconstruction, $*$ is the convolution operator, $4w$ is 4 times the 5×5 w matrix above, and $[G_{k+1}]_{\uparrow 2}$ represents an upsampling of the layer $k+1$ above to match the resolution of the current layer L_k of the RE Laplacian pyramid. If the formula has been applied enough times to successfully recover $\hat{G}_0$, it will be the reconstructed composite image. The upsampling (**expand** function) is represented by the following equation:

$$P_{l,k}(i,j) = 4 \sum_{m,n=-2}^{2} w(m,n) P_{l,k-1}(\frac{i+m}{2}, \frac{j+n}{2}) \tag{3.3}$$

where $P_{l,k}$ represents the expanded output level, w is the same 5×5 weight matrix defined above, $P_{l,k-1}$ represents the input level, and only integer indices contribute to the sum. The expand function is accomplished by padding every other column and row with zeros and then convolving the zero padded image with the weight matrix w. In affect, an upsampling by a factor of two is being accomplished.

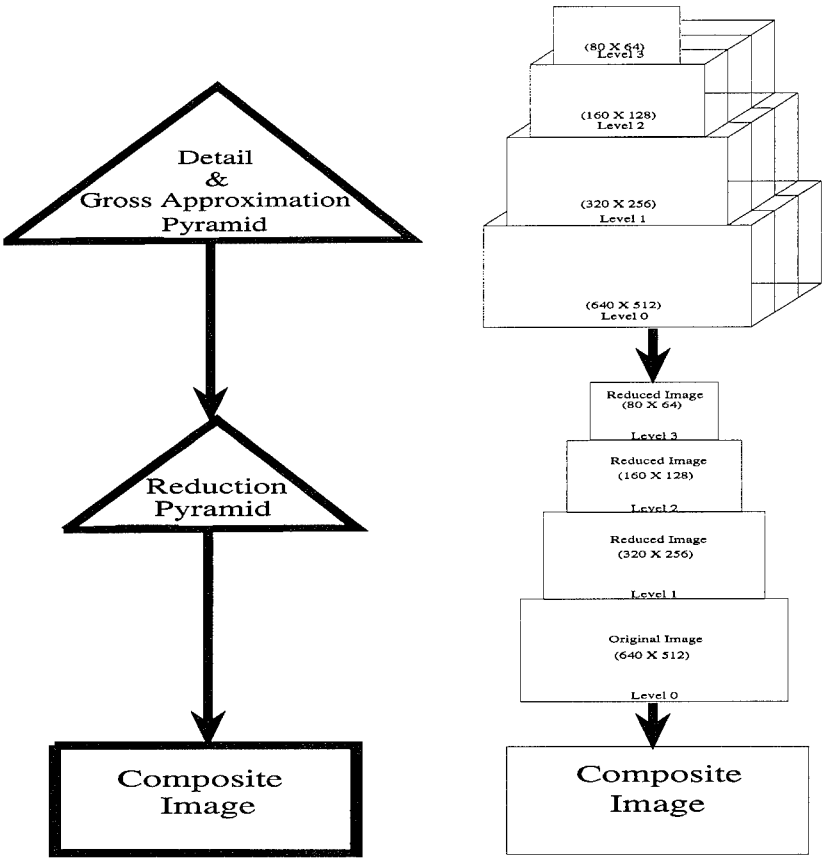


Figure 3.10 Reconstruction algorithm combines the gross approximation and details into a composite image.

3.3 Test Images

In order to evaluate the capabilities of the three fusion algorithms presented, multiple test images are generated with varying signal-to-noise ratios (SNR) and types of backgrounds. Random noise, with a uniform distribution on the interval (0.0,1.0), is scaled by a constant and is then added to various locations in the images. Where SNR is defined by:

$$SNR = log_{10} \frac{Energy\,in\,Signal}{Energy\,in\,Noise}$$

The energy in the signal is simply the sum of the squared pixel grey-scale values in the pure unmodified images of Lenna or Band 30. These are considered the "Golden" images for the test cases. The energy in the noise is computed by first subtracting the modified image from the unmodified image to get the noise. Then the sum of the squared difference provides the energy in the noise. Two different base images are used to obtain multiple types of backgrounds. The first image is the original (clean) version of Lenna [1] which contains areas of high frequencies, vertical/horizontal edges, low frequencies, and circular oriented details, left image in Figure 3.8. The second image is the image representation of band 30 from the AVIRIS hyperspectral sensor, Figure 4.9. Band 30 contains natural scenes (water, trees, fields) and urban structure (streets, runway, buildings). Correlated or uncorrelated noise was added to the test images in three different regions, thus creating sets of three images to be fused. The correlated noise was generated by convolving the uncorrelated noise with a low-pass filter. The level of correlation was computed as the full-width-half-max of the maximum amplitude of the autocorrelation of the convolution filter [7]. The low-pass filter used to generate the correlated noise was the same

$$5 \times 5$$

w filter used in Burt's fusion algorithm [5]. The full-width-half-max value was computed to be 4 pixels. Thus, each 4×4 neighborhood of noise is fully correlated. Figure 3.11 provides an example of three images containing uncorrelated noise with a SNR of 10db, within the area the noise was added. Figure 3.12 provides an example of three images containing correlated

Image Name	Correlated/Uncorellated	SNR of Affected Areas (db)	SNR of Overall Image (db)
Lenna_ 10db1	Uncorrelated	10	21.9
Lenna_ 10db2	Uncorrelated	10	23.9
Lenna_ 10db3	Uncorrelated	10	20.0
Lenna_ 5db1	Uncorrelated	5	16.9
Lenna_ 5db2	Uncorrelated	5	18.9
Lenna_ 5db3	Uncorrelated	5	15.0
Lenna_ 2.5db1	Uncorrelated	2.5	14.4
Lenna_ 2.5db2	Uncorrelated	2.5	16.3
Lenna_ 2.5db3	Uncorrelated	2.5	12.5
Lenna_ 5dbcorr1	correlated	5	17.1
Lenna_ 5dbcorr2	correlated	5	18.9
Lenna_ 5dbcorr3	correlated	5	15.1

Table 3.1 Signal-to-Noise Ratios for the Lenna Modified Test Images.

Image Name	Correlated/Uncorellated	SNR of Affected Areas (db)	SNR of Overall Image (db)
band30_ 5dbcorr1	correlated	5	21.1
band30_ 5dbcorr2	correlated	5	19.8
band30_ 5dbcorr3	correlated	5	15.9

Table 3.2 Signal-to-Noise Ratios for the Band 30 Modified Test Images.

noise with a SNR of 10db, within the area the noise was added. Table 3.1 provides a listing of the SNRs for all of the test images using Lenna. Table 3.2 provides a listing of the SNRs for the test images using Band30. The tables include a column for the overall image SNR to provide for a comparison to the fused image results, since there will be reconstruction error to consider.

3.4 Conclusion

This chapter defined an image fusion algorithm which was based upon the visual perception of the human analyst. The fusion algorithm uses contrast sensitivity to determine the salient parts of the input images to be fused. The method of generating test images to evaluate the three fusion algorithms was also presented. The test images are generated by adding uncorrelated or correlated noise to images with various backgrounds. The next chapter

Figure 3.11 Three Test Images of Lenna with 5db energy SNR of uncorrelated noise added at three different locations.

Figure 3.12 Three Test Images of Lenna with 5db energy SNR of correlated noise added at three different locations.

discusses the results of image fusion using the three fusion algorithms presented in chapters three and four.

IV. Results

4.1 Introduction

This chapter provides examples of fusion that employ all three of the image fusion techniques previously described in chapters two and three. The first example fuses the test images described in section 3.3 to illustrate the capabilities and limitations of the different methods. The second example combines three SAR images to evaluate how fusion affects the automated target recognition of the scenes. The third example integrates three bands of AVIRIS data to provide a pictorial representation of IR and Visual data fusion from a single sensor. A fourth example employs only the algorithm developed in this thesis to fuse nine bands from the AVIRIS sensor.

4.2 Hierarchical Image Fusion of Test Images

In this section, the results of fusing the test images described in section 3.3 are presented. The test images were fused using the algorithms by Burt [5] and Toet [2, 18, 19] described in chapter two and the method developed in this thesis which was detailed in chapter three.

All of the fusion performed in this thesis was implemented using the recommended parameters from Burt [5] and Toet [2, 18, 19]. The parameters were the same for every fusion example regardless of the type of data fused (AVIRIS, SAR or test images). Neither Burt [5] nor Toet [2, 18, 19] stated in their articles that their algorithm's parameters were problem dependent, therefore no attempt was made to alter the parameters to optimize a fusion result. The parameters used for the fusion method developed here were also the same for all fusion examples. The parameters for each algorithm are as follows:

Burt's fusion algorithm was implemented using the following parameters:

1. Six layers of decomposition (i.e., 5 levels of details) using Burt's recommended w matrix and $d1$ thru $d4$ filters [5].
2. A 3×3 p matrix of all ones.

3. An alpha of 0.9.

Alexander Toet's fusion algorithm was implemented using six layers of decomposition and Toet's recommended weight matrix w [2, 18, 19].

The method developed in this thesis was implemented using the following parameters:

1. Six layers of decomposition (i.e., 5 levels of details) using Burt's recommended w matrix [5]
2. A window size of 40×40.
3. A shift of 8.
4. A threshold of 0.05.

Three sets of results are presented here for comparison. One shows the results of fusion using uncorrelated noise and the other two using correlated noise and different backgrounds. The remaining fusion results are shown in Appendix C. Figure 4.1 represents the results of fusing the first three images defined in Table 3.1. Figure 4.2 represents the results of fusing the three images defined in Table 3.1 that use correlated noise. Figure 4.3 represents the results of fusing the three images defined in Table 3.2 that use correlated noise.

Three composite images were created for visual reference to aid in analysis of the fusion results, Figures 4.4, 4.5, 4.6. The composite images were created by inserting the noise from each input image into their respective locations in the original image of Lenna or Band 30, respectively. It can be seen by comparing the fused images in Figure 4.1 with the reference images of Figure 4.4, and by looking at the fusion results in Table 4.1 that the method developed in this thesis does a better job of de-emphasizing uncorrelated noise in the input images than either Burt's or Toet's methods. It can also be seen by comparing the fused images in Figures 4.2 and 4.3 with the reference images of Figures 4.5 and 4.6 and by looking at the fusion results in Table 4.2 that the method developed here also does a better job of de-emphasizing correlated noise in the input images. In most cases, see Table 4.1 , the method developed in this thesis even increased the SNR of the fused results above that of any input image. However, even though the fused SNR was better than either Burt's or Toet's results,

Figure 4.1 Fusion results of test images are arranged as follows: Burt top left, Toet top right, and Contrast sensitivity bottom. The input images, which contain uncorrelated noise, are the first three images defined in Table 3.1

Figure 4.2 Fusion results of test images are arranged as follows: Burt top left, Toet top right, and Contrast sensitivity, bottom. The input images are the three images defined in Table 3.1 that contain correlated noise.

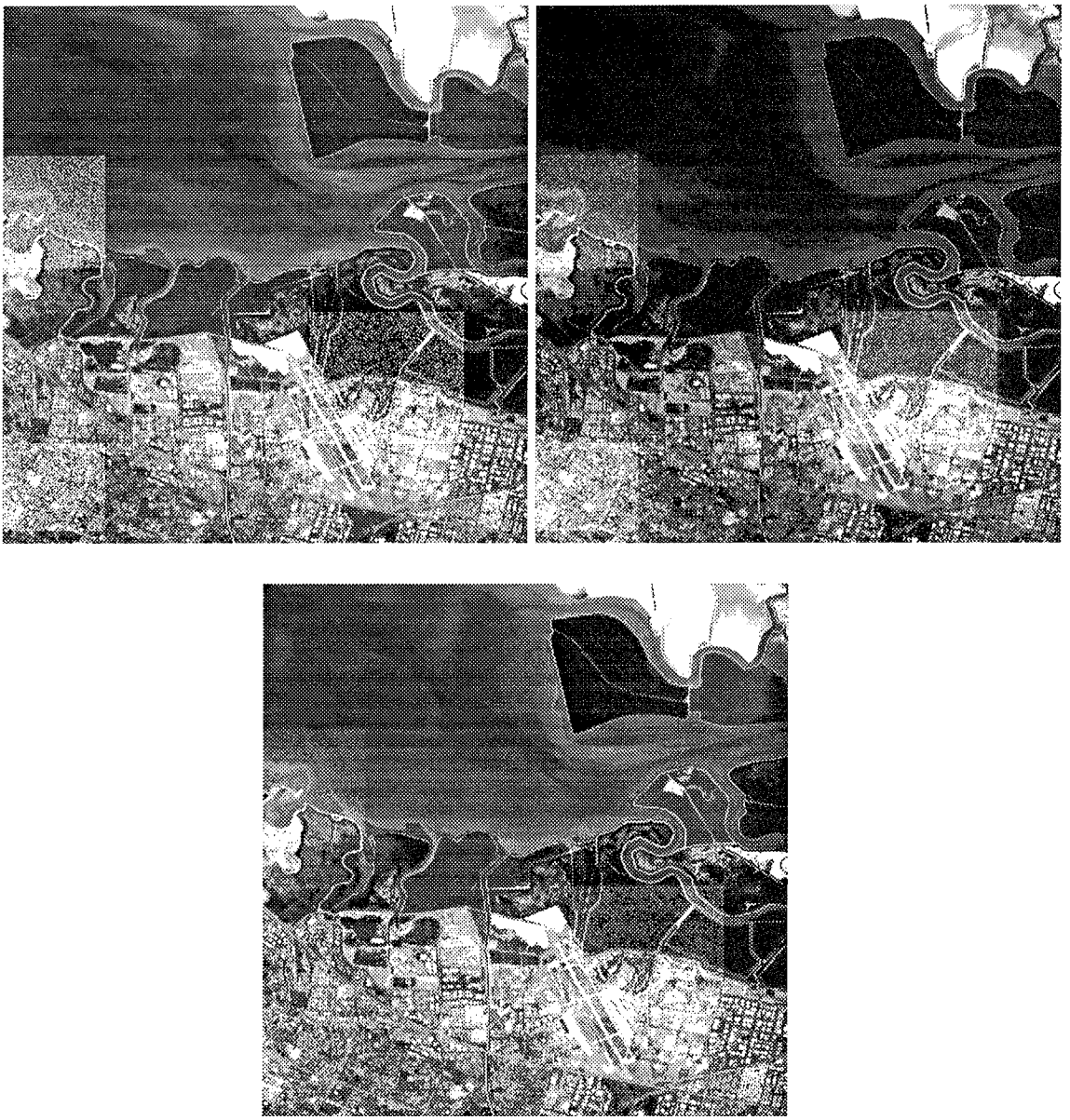


Figure 4.3 Fusion results of test images are arranged as follows: Burt top left, Toet top right, and Contrast sensitivity, bottom. The input images are the three images defined in Table 3.2 that contain correlated noise.

the fusion method developed here did not improve the SNR of the fused Band 30 images over the SNR of the Band 30 input images. Two possible explanations for the differences in the results of that for the Lenna noise images and the Band 30 image are:

1. The frequency of the added noise and the spatial content for some parts of the Band 30 images were such that when combined they provided enough energy in the perceptual domain to make that area more salient.
2. The reconstruction error was larger due to the higher energy content of the Band 30 images, thus creating a larger difference between the fused composite and the ideal Band 30 image.

It is also important to note that just because two images differ in Euclidean space, they may not be perceived as different. How well changes to local contrast can be perceived is a function of the contrast sensitivity [17, 12, 10].

Comparing Burt's method to Toet's shows that Burt's method provided better noise reduction, but it still weighted the noisy parts of the images as more salient than the non-noisy parts. It can also be seen from the fusion results that Toet's method selected the high energy noisy parts of the input images to retain in the composite, even though the lower frequency parts of the input images contained the information desired. This loss of information in the input images, due to selecting the high energy noisy parts of the input images clearly demonstrates a limitation to both Toet's and Burt's fusion methods, see sections 2.8 and 2.15. The noise sensitivity of Toet's and Burt's methods is clearly overcome using the method proposed in this thesis.

4.3 Hierarchical Image Fusion of Synthetic Aperture Radar (SAR) Image Data

In this section, the results of fusing three separate SAR images are presented. Fusion of SAR data is presented as one example of how fusion may affect ATR methods and that the fusion algorithms are not strictly limited to hyperspectral data. It was not intended to provide a Stochastic analysis of how fusion affects ATR.

Figure 4.4 Original image of Lenna and a composite image where each area of noise contains a 10 db SNR of uncorrelated noise added.

Figure 4.5 Original image of Lenna and a composite image where each area of noise contains a 10 db SNR of correlated noise added.

Figure 4.6 Original image of Band 30 and a composite image where each area of noise contains a 10 db SNR of correlated noise added.

Image Method	Image Group	SNR of Overall Image (db)
Burt	Lenna_ 10db	20.0
Toet	Lenna_ 10db	14.6
Contrast Sensitivity	Lenna_ 10db	24.2
Burt	Lenna_ 5db	14.5
Toet	Lenna_ 5db	9.5
Contrast Sensitivity	Lenna_ 5db	19.6
Burt	Lenna_ 2.5db	11.7
Toet	Lenna_ 2.5db	6.9
Contrast Sensitivity	Lenna_ 2.5db	17.1

Table 4.1 Signal-to-Noise Ratios for the Fused Test Images Containing Uncorrelated Noise.

Image Method	Image Group	SNR of Overall Image (db)
Burt	Lenna_ 5dbcorr	15.5
Toet	Lenna_ 5dbcorr	9.9
Contrast Sensitivity	Lenna_ 10dbcorr	19.6
Burt	band30_ 5dbcorr	16.0
Toet	band30_ 5dbcorr	11.2
Contrast Sensitivity	band30_ 5dbcorr	20.0

Table 4.2 Signal-to-Noise Ratios for the Fused Test Images Containing Correlated Noise.

Input Image	Hits	Misses	False Alarms
KM15hh	4	2	7
KM15hv	3	3	4
KM15vv	3	3	4

Table 4.3 Results of Automated Target Recognition of Individual SAR Images.

The SAR images are of the same scene and radar angle, but with three different polarimetric filter combinations. A target recognition algorithm was applied to the individual input images and then to the fused images to analyze the effects of fusion on automated target recognition. The three SAR images were fused using the same three methods described in section 4.2. Figure 4.7 represents the three SAR input images.

The target recognition algorithm used to analyze the images declares a pixel as a target if its intensity value is at least one standard deviation above the mean of the image. A neighborhood of 10×10 is used to mark the location of possible targets. Thus, for every 10×10 neighborhood of pixels, if a single pixel is determined to be a target an x will be placed over the individual pixel. If more than one pixel within the 10×10 box is determined to be a target, the $\times$ is placed in the center of the possible targets. The data has been ground-truthed and all possible targets in the scene are marked with a plus sign. Therefore, when an x is placed in the same spot as a plus sign this is called a hit, meaning a correct classification has been made. If an $\times$ is placed somewhere other than over a plus sign a false classification or false positive has occurred. If a plus sign has no $\times$ placed over it then a miss or false negative has occurred. Figure 4.7 also shows the results of the target recognition analysis of the individual input images by the placement of the $\times$'s and $+$'s. Figure 4.8 show the results of image fusion and target recognition analysis of the fused SAR images. Table 4.3 provides a tally of the target recognition results using the individual images and Table 4.4 provides the target recognition results of the fused data.

Burt's fusion method obtained the best results for the automated target recognition. It maintained the best target recognition results obtained using the individual images alone of 4 hits and 2 misses while lowering the false alarm rate by 3. Toet's method obtained the same

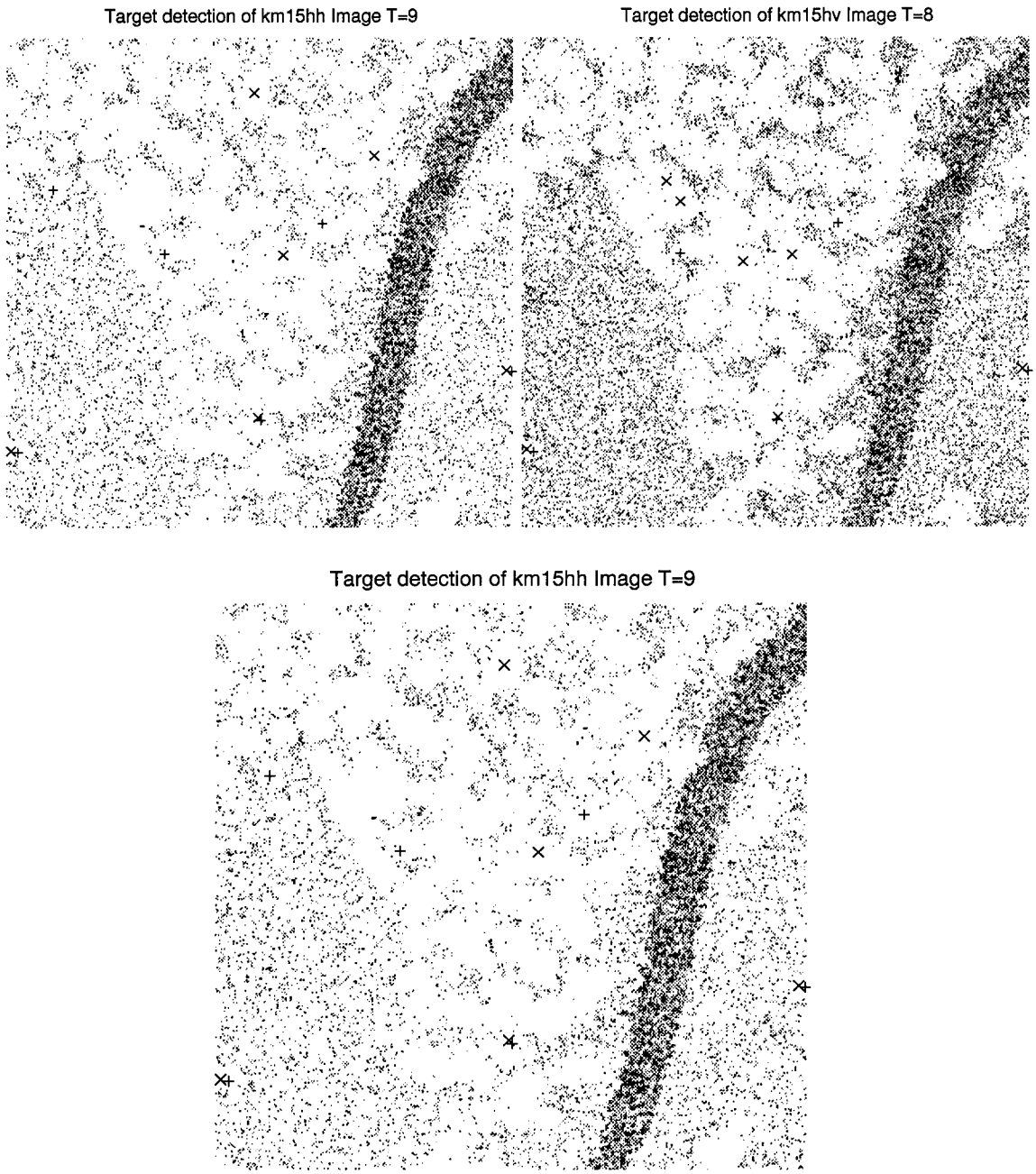


Figure 4.7 SAR images with polarimetric orientations horizontal/horizontal, horizontal/vertical, and vertical/vertical are located at top left, top right, and bottom, respectively. An + indicates an actual target and an × indicates where the ATR algorithm says a target is.

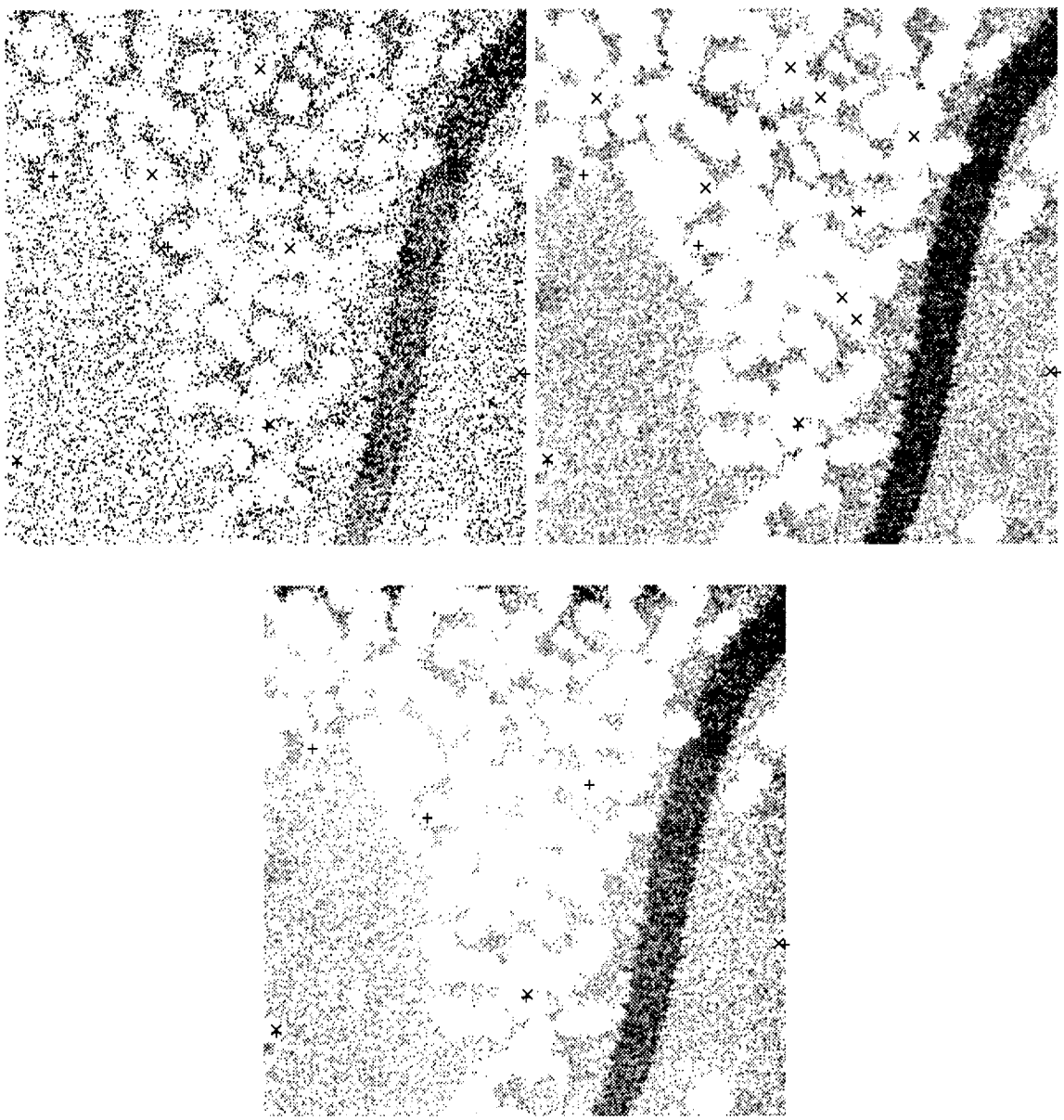


Figure 4.8 Fusion results of SAR images are arranged as follows: Burt top left, Toet top right, and Contrast sensitivity bottom. An + indicates an actual target and an × indicates where the ATR algorithm says a target is.

Input Image	Hits	Misses	False Alarms
Burt	4	2	4
Toet	4	2	7
Contrast sensitivity	3	3	0

Table 4.4 Results of Automated Target Recognition of Fused SAR Images.

results as the best individual image case. The fusion algorithm developed in this thesis did not achieve the same number of hits as the best of the individual case, however, it did improve upon the accuracy of two of the three input images by lowering the false alarm rate to 0. Thus, demonstrating that while fusion based upon contrast sensitivity does a good job for human analysis, it doesn't necessarily do a good job for automated target recognition.

4.4 Hierarchical Image Fusion of IR and Visual Bands of AVIRIS Image Data

In this section, the results of fusing three bands of image data from the AVIRIS hyper-spectral sensor are presented. The three bands include one image from the visual frequency range and two from the infrared. Specifically, they are band numbers 30, 60, and 90, which are in the 0.67μm to .68μm, the 0.94μm to .95μm, and the 1.22μm to 1.23μm band-pass ranges, respectively. Again, fusion was performed using the same methods as in section 4.2. Figures 4.9, 4.10, and 4.11, represent the three AVIRIS input images.

It can be seen by comparing the fused images in Figures 4.12, 4.13, and 4.14 with the input images in Figures 4.9, 4.10, and 4.11 that all three methods do a good job of preserving visual information from each input image. However, comparing the three fused results to each other, Toet's fusion method does not have the same amount of detail that is present in the other two. For example, looking at the upper right hand corner of the fused images, it is easy to see that Toet's method causes the details to become washed out. In order to make the details in the upper corner more distinguishable, the overall intensity level of the image has to be reduced. The reduction in intensity then causes the other areas of the image to become less distinct. Also the runway and surrounding area is more clear in Burt's method and the contrast sensitivity method than it is in Toet's fusion method.

Comparing the fusion results of the method developed in this thesis with Burt's shows that they both have similar characteristics. Each method appears to preserve features in the input images that are dominant. An example of this preservation is observed by looking at the road that is a dominant feature in bands 60 and 90 but not in band 30. The road is located in the lower third of the image and extends from the left edge of the image all the way to the right.

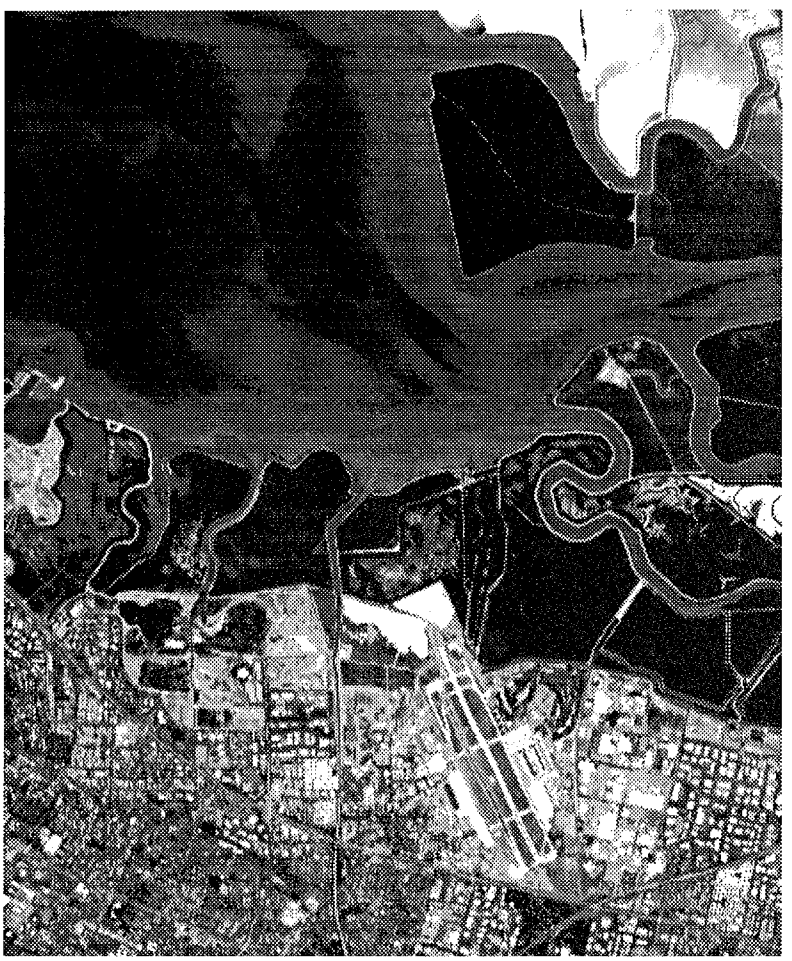


Figure 4.9 AVIRIS image representing band 30

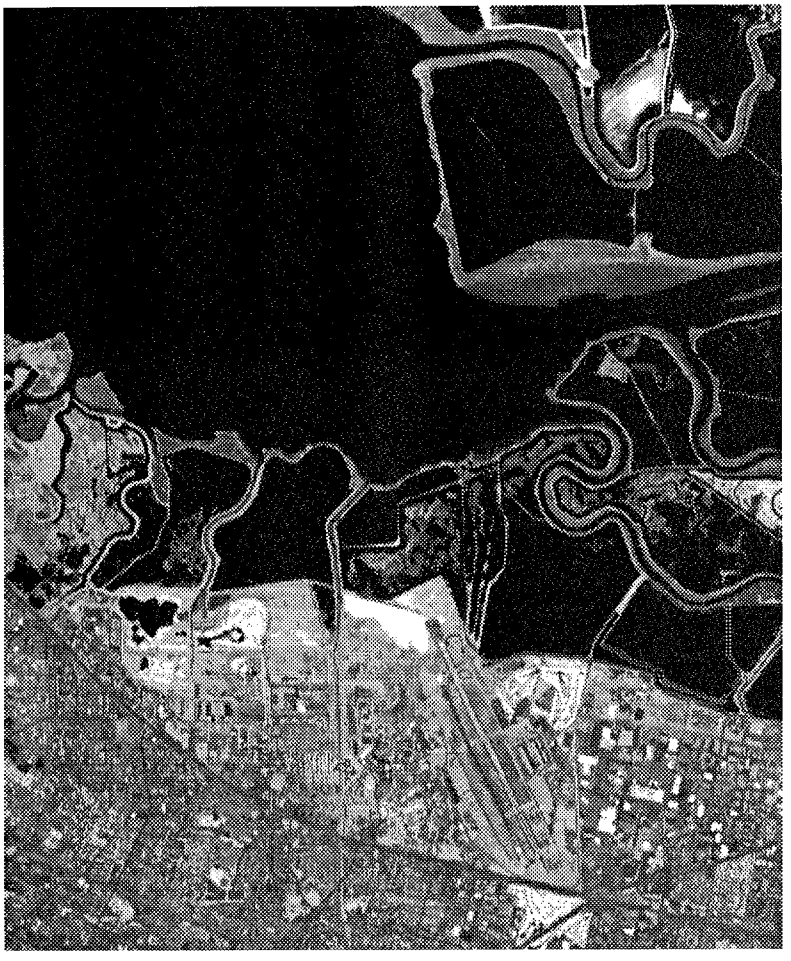


Figure 4.10 AVIRIS image representing band 60

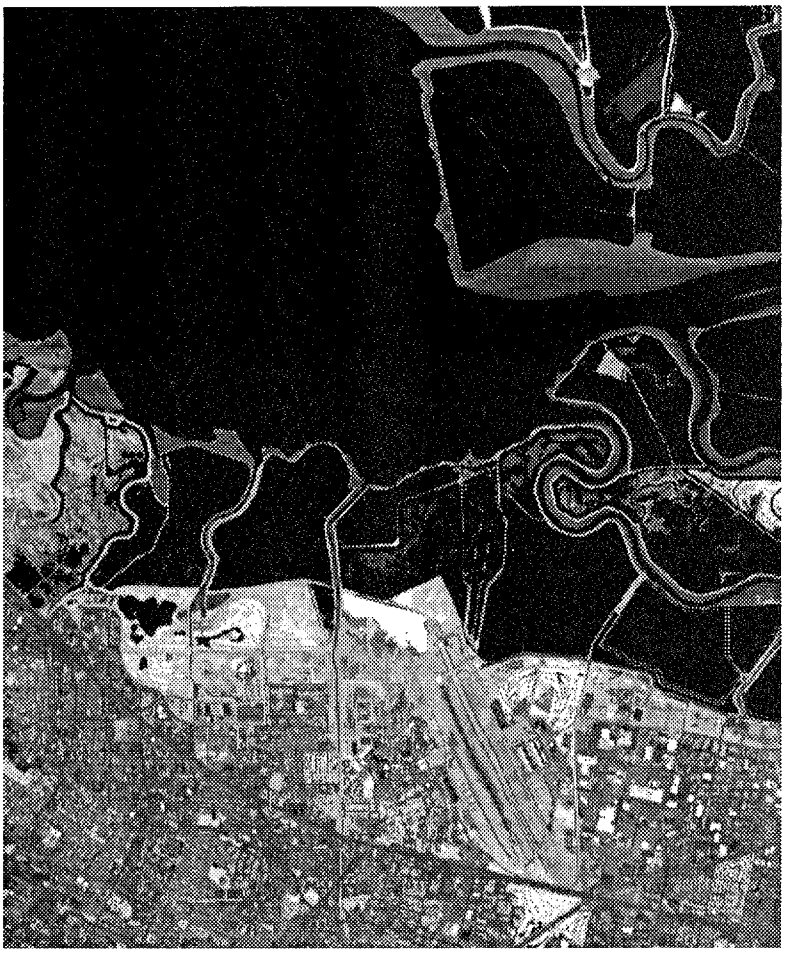


Figure 4.11 AVIRIS image representing band 90

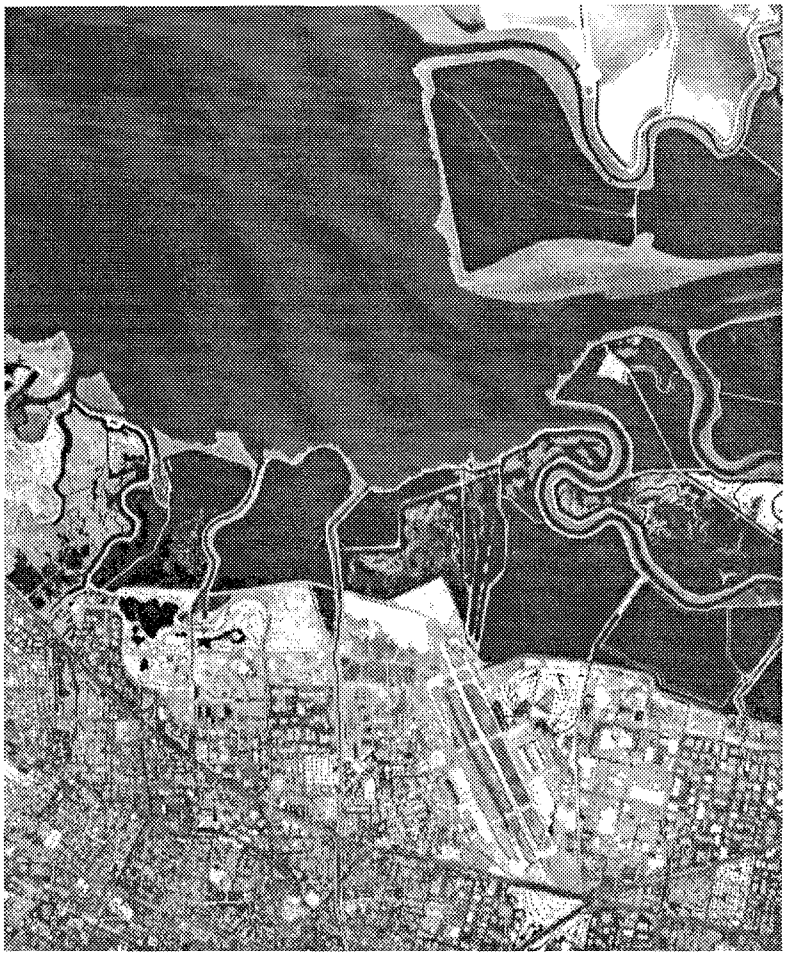


Figure 4.12 Burt fusion results of AVIRIS bands 30, 60, and 90.

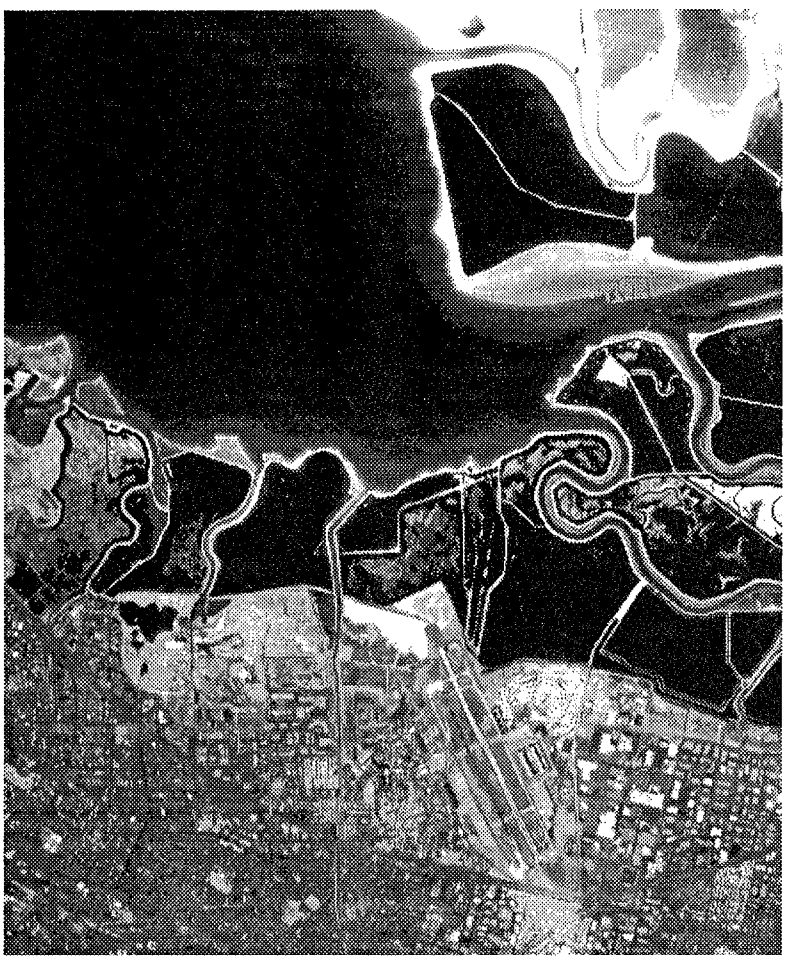


Figure 4.13 Toet fusion results of AVIRIS bands 30, 60, and 90.

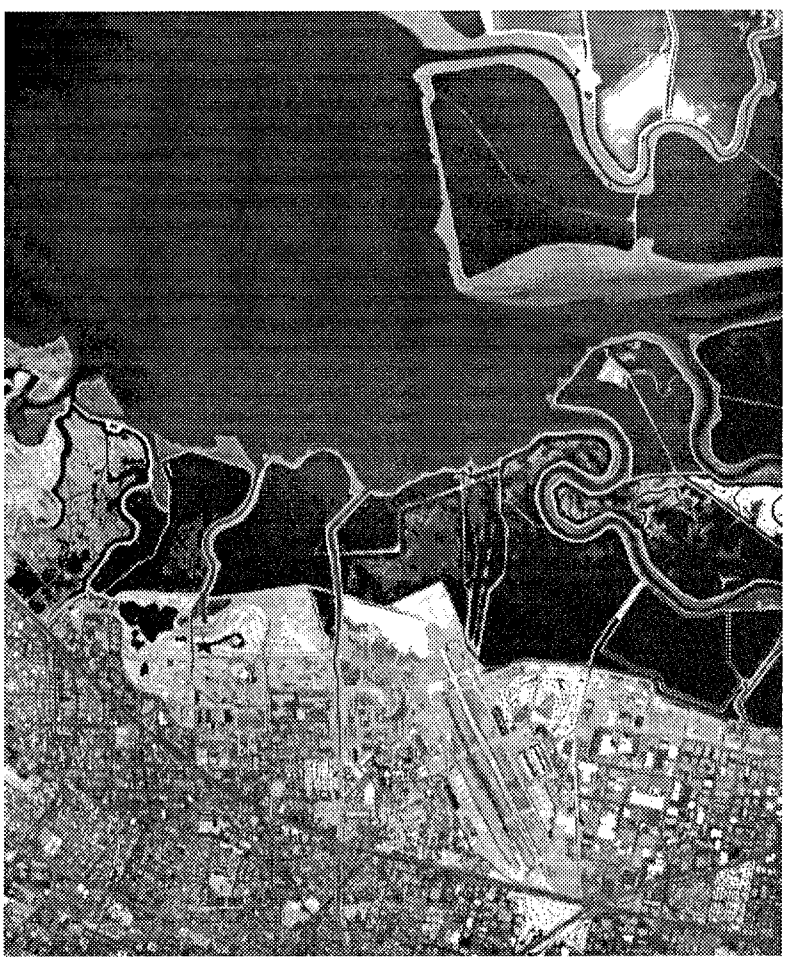


Figure 4.14 Contrast sensitivity fusion results of AVIRIS bands 30, 60, and 90.

It can clearly be seen as a continuous road in bands 60 and 90, but it is hard to distinguish in band 30. Looking at the fused results in figures 4.12 and 4.14 it can be seen that the road is preserved in the composite.

An example of how fusion of the input bands provides better detail in the composite than in the individual input images alone is also shown in the fused images of all three methods. The airport that can be seen in the lower right quadrant of the input images has been combined in the composite image to provide more detail than was in all three of the input images separately.

4.5 Hierarchical Image Fusion of 10 Bands of AVIRIS Image Data

This section is intended to provide an example of how the fusion method developed in this thesis can successfully fuse many bands from the AVIRIS sensor while causing no loss of information nor a loss of dynamic range.

The input images are from bands 30 through 40 from the AVIRIS hyperspectral sensor. The combined band-pass range of the images is from 0.67μm to .75μm. The Figures representing the input images are displayed in Appendix B. Figure 4.15 represents the results of fusion.

4.6 Conclusion

This chapter presented the results of hierarchical image fusion using two methods presented by Peter Burt [5] and Alexander Toet [2, 18, 19], and a third method which was developed in this thesis. The fusion algorithms were tested using test images defined in the previous chapter, SAR images with different polarimetric orientations (to evaluate the effect of fusion on automated target recognition), and images from the AVIRIS hyperspectral sensor. The results showed that contrast sensitivity based fusion was superior in noise reduction than either Burt's [5] or Toet's [2, 18, 19] methods no matter what type (correlated/ uncorrelated) or level of noise was added to the images. It was also shown that contrast sensitivity based fusion could successfully fuse many bands from the AVIRIS hyperspectral sensor. An example that

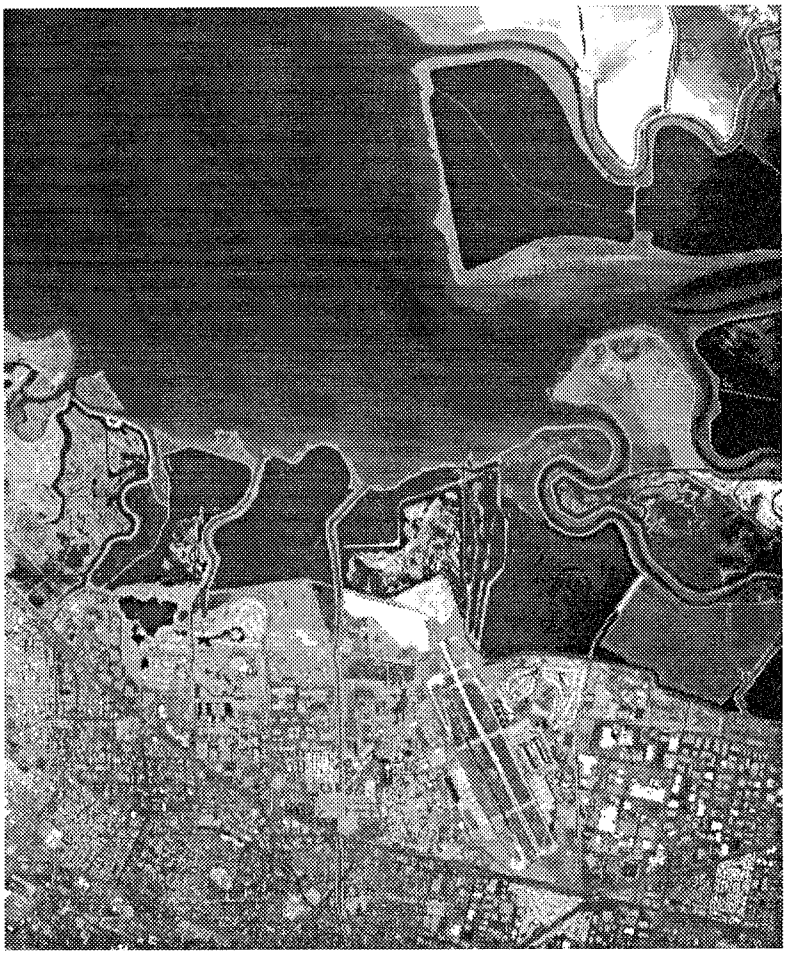


Figure 4.15 Contrast Sensitivity Fusion results of AVIRIS bands 30 through 40, Excluding 33.

fused ten bands was given. The next chapter will provide a discussion on the conclusions of this research and some recommendations for follow-on research.

V. Conclusions and Recommendations

The conclusions and recommendations presented in this chapter are based upon the results detailed in the previous chapter. First, conclusions derived from the results of fusing the test images in chapter four is given. Second, a discussion on the affects that fusion may have on automated target recognition of Synthetic Aperture Radar data is provided. Third, an analysis of the fusion of AVIRIS data is presented. Fourth, general conclusions about all three of the image fusion techniques used in this research are detailed. Finally, some follow-on research recommendations are given.

5.1 Contrast Sensitivity Fusion

Referring to Figures 4.1, 4.2, and 4.3, image fusion using contrast sensitivity as a salience measure produces fused images that are visually better than either Burt's [5] or Toet's [2, 18, 19] methods. In every instance, the amount of noise that is retained in the composite images is lower for the contrast sensitivity method than it is for either of the other two methods.

The limits to Burt's and Toet's methods which were discussed in sections 2.15 and 2.8, respectively, are clearly displayed in the results. Again, referring to Figures 4.1, 4.2, and 4.3, Toet's method definitely emphasizes the noise in the input images and Burt's approach selects the higher energy in the noise as most salient. It is also shown in Tables 3.1 and 3.2 that contrast sensitivity based fusion provides better SNRs over a wide range of noises and backgrounds, and that the results are independent of the type (correlated vs uncorrelated) and strength of the added noise. Also, as the noise in the images is increased, the performance of the contrast sensitivity based fusion actually improved over the other methods.

It is important to note that SNR analysis alone is not enough to judge a good fusion result. Some of the noise may be in a less "contrast sensitive" region of the visual system and it would not be visually noticeable. Therefore, even though an image may have a better SNR than another image, a visual inspection of the fused images must confirm the results.

5.2 *Synthetic Aperture RADAR (SAR) Image Fusion and Automated Targeting*

The fusion of SAR data was only provided as an example to show that the fusion methods developed and investigated in this thesis are not strictly limited to hyperspectral data. This was not an attempt to provide a stochastic analysis about how fusion impacts ATR of SAR data.

The results of target recognition of the SAR data, after fusion, indicates that image fusion based upon contrast sensitivity, while shown to be good for visual analysis, may not be the best approach when the recognition algorithm uses thresholding as a criteria, see Tables 4.3 and 4.4. However, the results do indicate that the fusion algorithms proposed by Burt and Toet are possibly viable means of combining the different polarimetric images to aid in target recognition. In the worst case, Toet's method maintained the best results of target recognition using the input images separately. In the best case, image fusion using Burt's method maintained the best number of correct classification and at the same time reduced the number of false alarms by half, again refer to Tables 4.3 and 4.4.

5.3 *AVIRIS Hyperspectral Data Fusion*

Referring to the results of chapter four, contrast sensitivity based fusion successfully fuses image data from the AVIRIS hyperspectral sensor and still maintains the relevant visual information in the input scenes. Figures 4.14 and 4.15 clearly demonstrate that images from multiple spectral bands can be fused to reduce image storage and increase the amount of information in a given scene, as was discussed in the previous chapter. Even though the contrast sensitivity method provided the best results, Burt's and Toet's methods were also capable of fusing AVIRIS data with some limitations. One important limitation is that both methods considered noise in the images to be very important. Thus, creating lower SNRs in the fused images than the contrast sensitivity method. Another limitation, that Toet's method experienced, was a reconstruction halo or edge effect, Figure 4.3. This reconstruction effect was caused by selecting the maximum point in each ratio pyramid, then the point's value was propagated into a wider and wider area as the images were reconstructed.

5.4 *Overall Conclusions*

Toet's [2, 18, 19] maximum contrast fusion method is less computationally intense than either of the other two methods discussed, but it is more susceptible to noise. Burt's [5] match and saliency fusion method is less susceptible to noise than Toet's method, but it suffers from a need to define an arbitrary weight matrix p to determine saliency. No method of determining an optimal p matrix was provided by Burt or found during the implementation of Burt's algorithm during this research. Finally, the new multi-resolution fusion algorithm, which uses contrast sensitivity as a salience criteria, has been demonstrated to be a successful method of image fusion. It works well on hyperspectral image data, is resistant to image noise, and provides better SNRs and aesthetic quality than either Burt"s [5] or Toet's [2, 18, 19] fusion methods.

5.5 *Recommendations for Follow-On Research*

There are several areas of interest that still need to be explored. For example, six levels of decomposition were used in the fusion methods implemented in this thesis, but an analysis of how many levels are actually needed for a given scene still needs to be investigated.

Also, the decomposition and reconstruction methods implemented in the contrast sensitivity approach had reconstruction errors of about 2%. A method of deconstruction and reconstruction that minimizes the error is still needed.

Further study is also recommended to analyze the results of using an individuals contrast sensitivity response to evaluate the improvement to the image fusion algorithm. It would be useful as well to research the benefits of using suprathreshold contrast as the contrast response of the human visual system.

Finally, a wavelet decomposition method that functionally relates directly to the human contrast sensitivity response would be helpful because it would eliminate the need for the Fourier analysis. The energy in the detail coefficients could be compared directly as with Toet's method, because now the energy domain has been transformed to the perceptual domain.

Appendix A. Test Images

This Appendix provides a figure for every test image used in this thesis.

Figure A.1 Three Test Images of Lenna with 10db SNR of uncorrelated noise

Figure A.2 Three Test Images of Lenna with 5db SNR of uncorrelated noise

Figure A.3 Three Test Images of Lenna with 2.5db SNR of uncorrelated noise

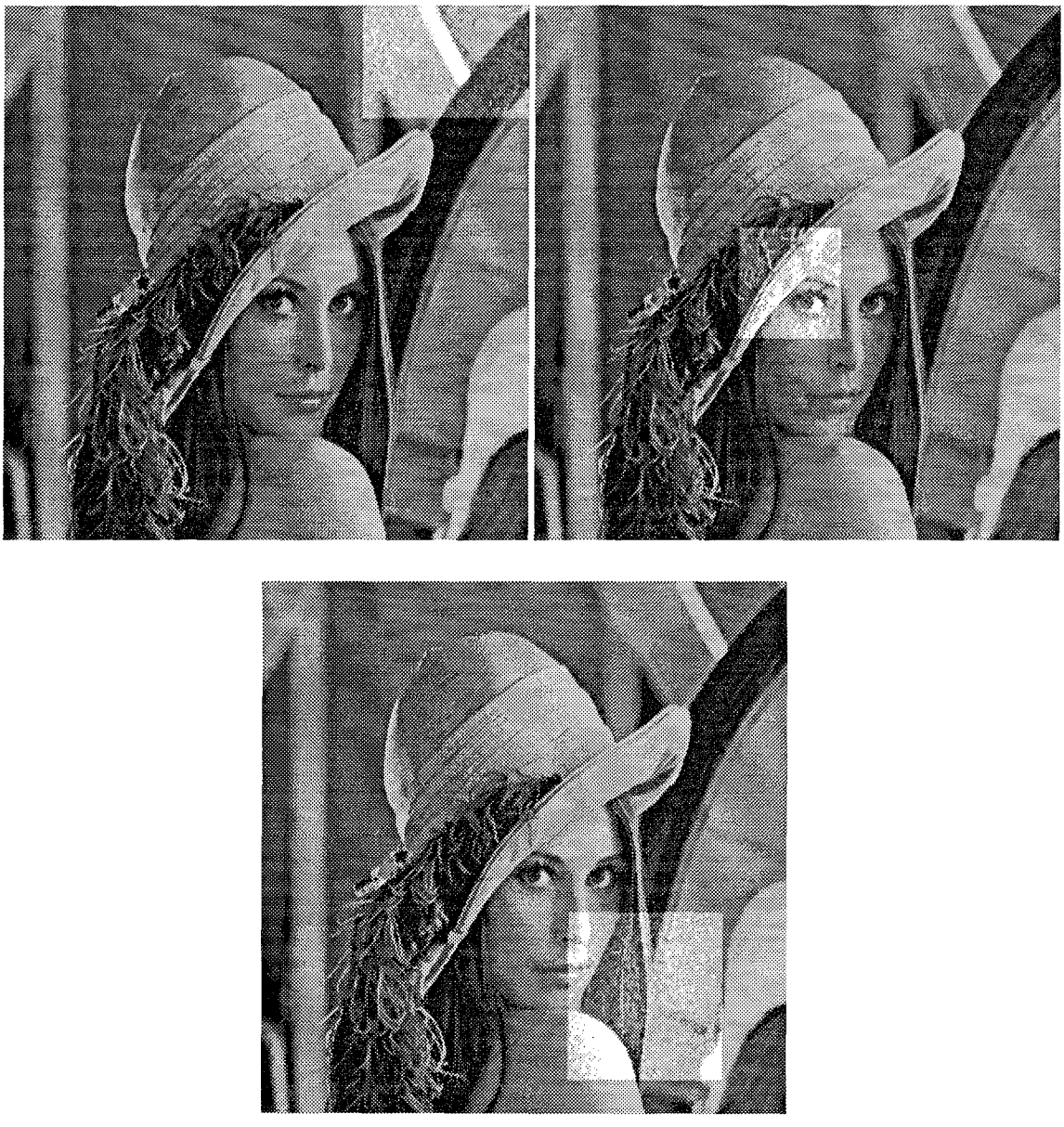


Figure A.4 Three Test Images of Lenna with 10db SNR of correlated noise

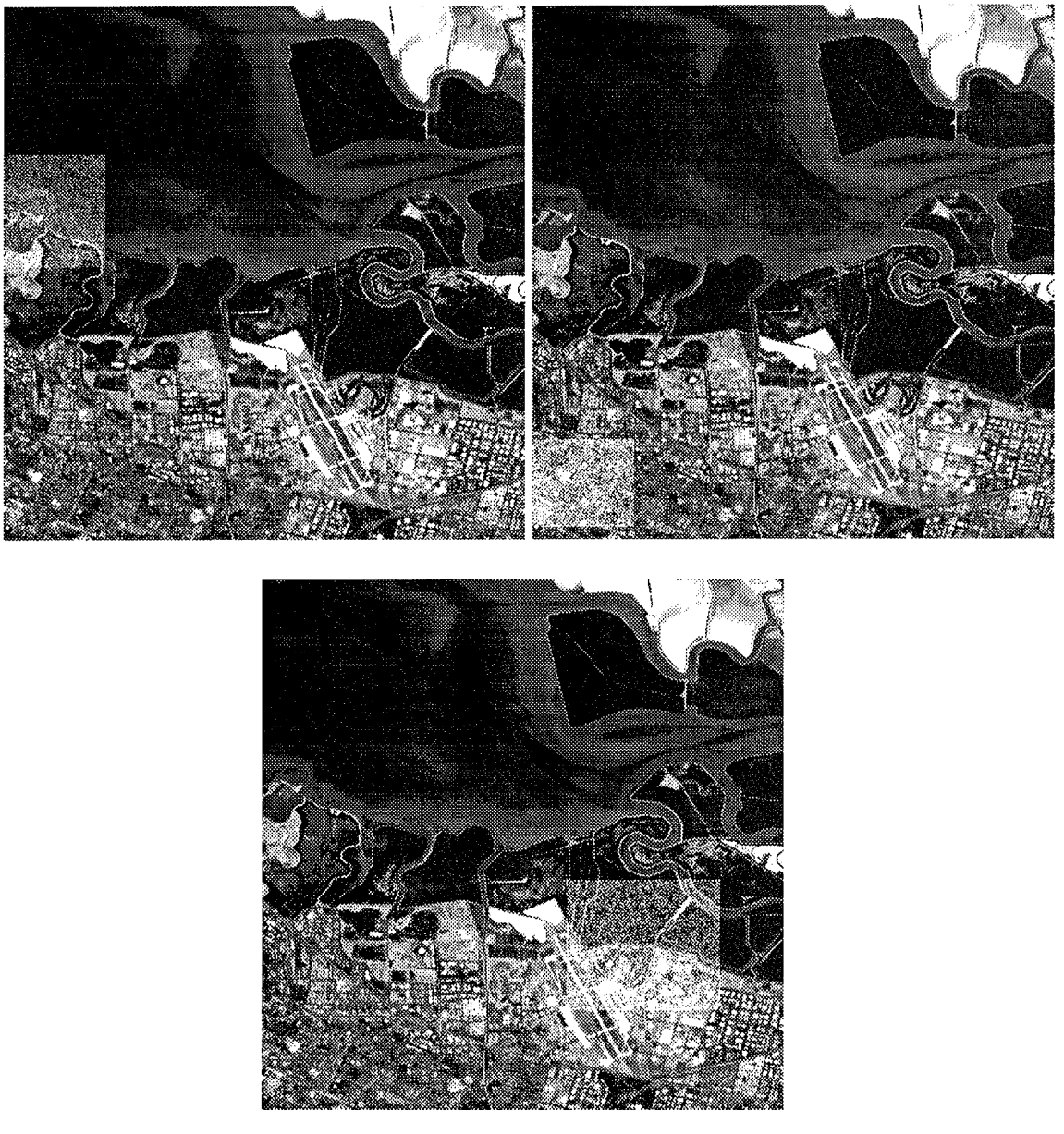


Figure A.5 Three Test Images of Band 30 with 5db SNR of uncorrelated noise

Appendix B. AVIRIS Hyperspectral Images

This Appendix provides a figure for every AVIRIS image used in this thesis.

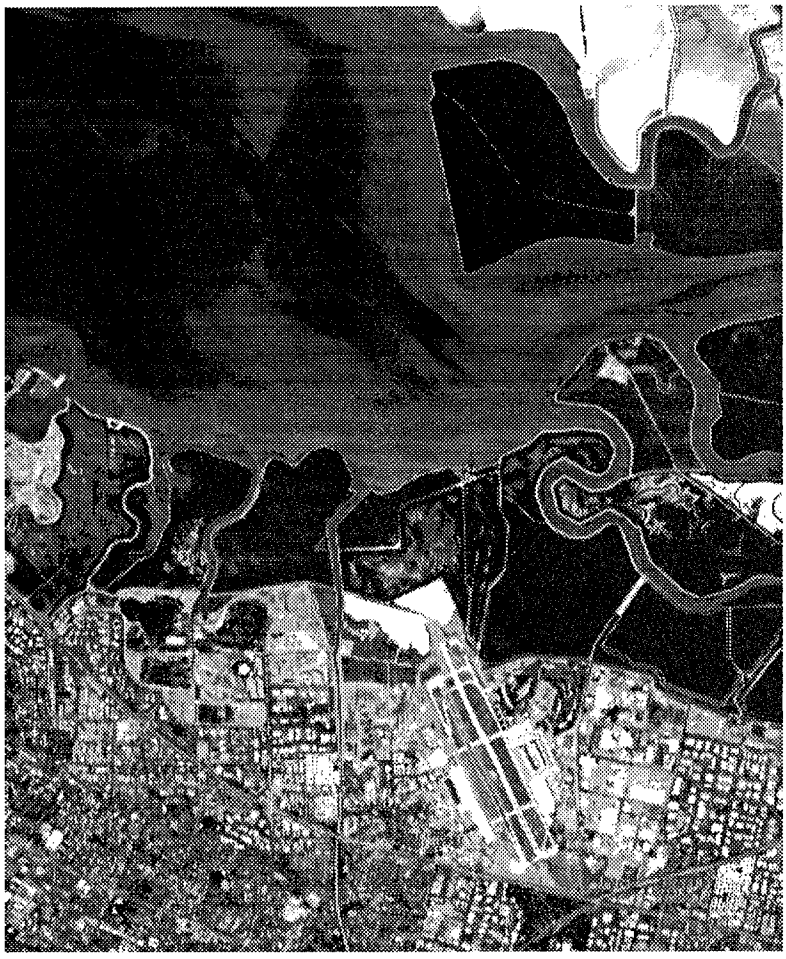


Figure B.1 AVIRIS Hyperspectral Image Representing Band 30

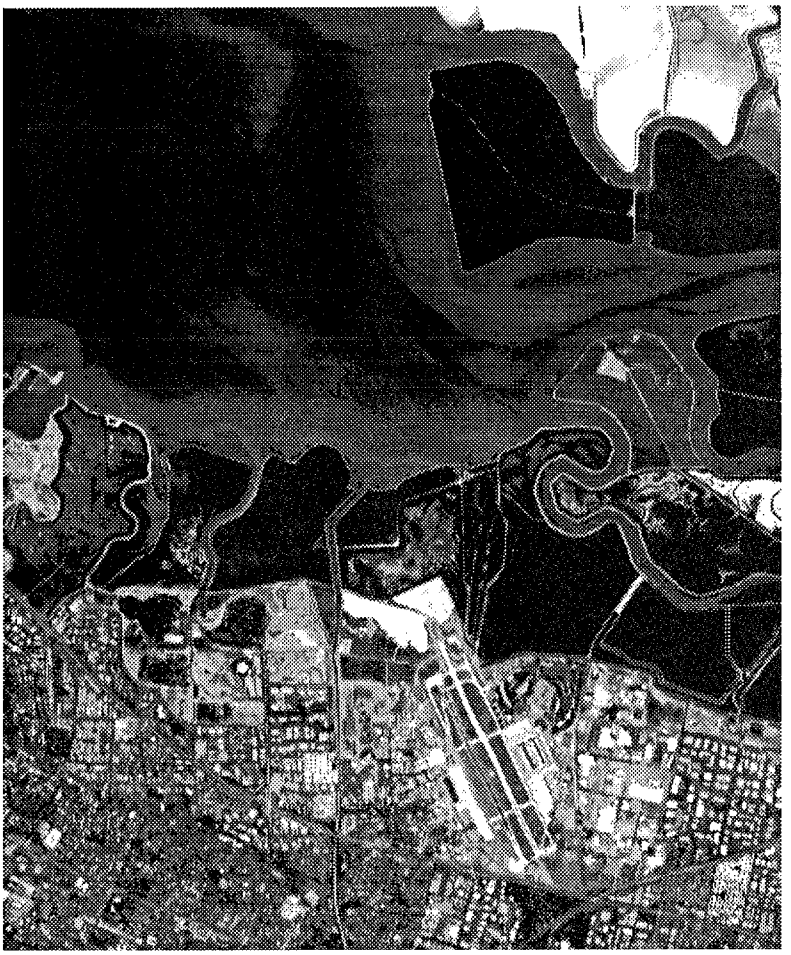


Figure B.2 AVIRIS Hyperspectral Image Representing Band 30

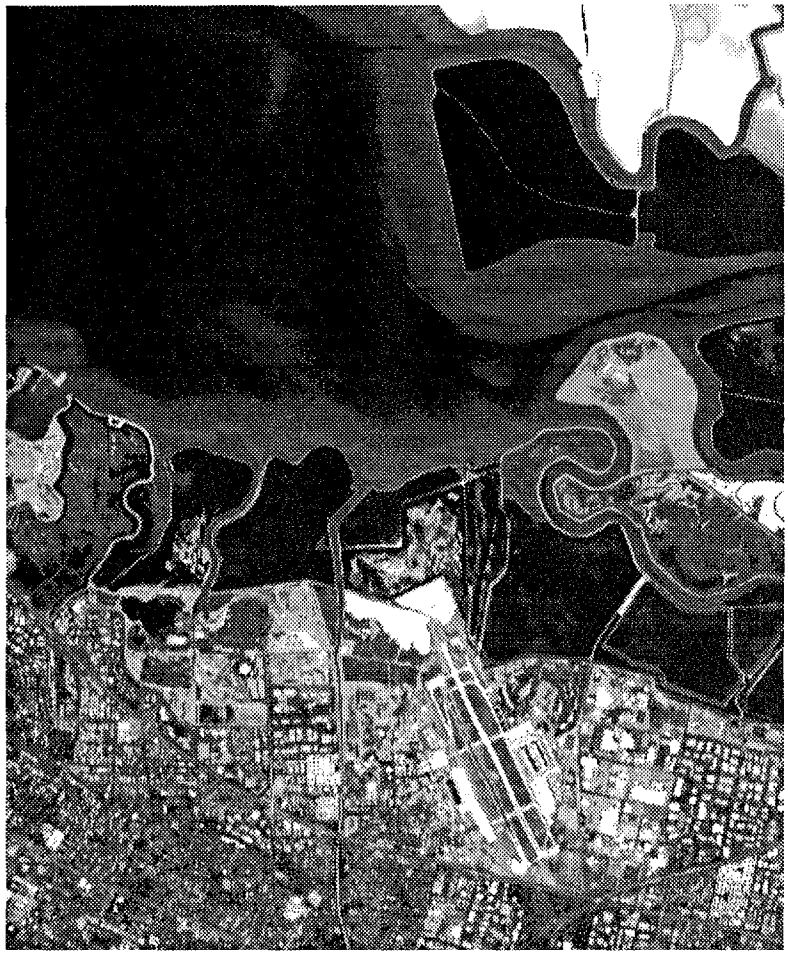


Figure B.3 AVIRIS Hyperspectral Image Representing Band 32

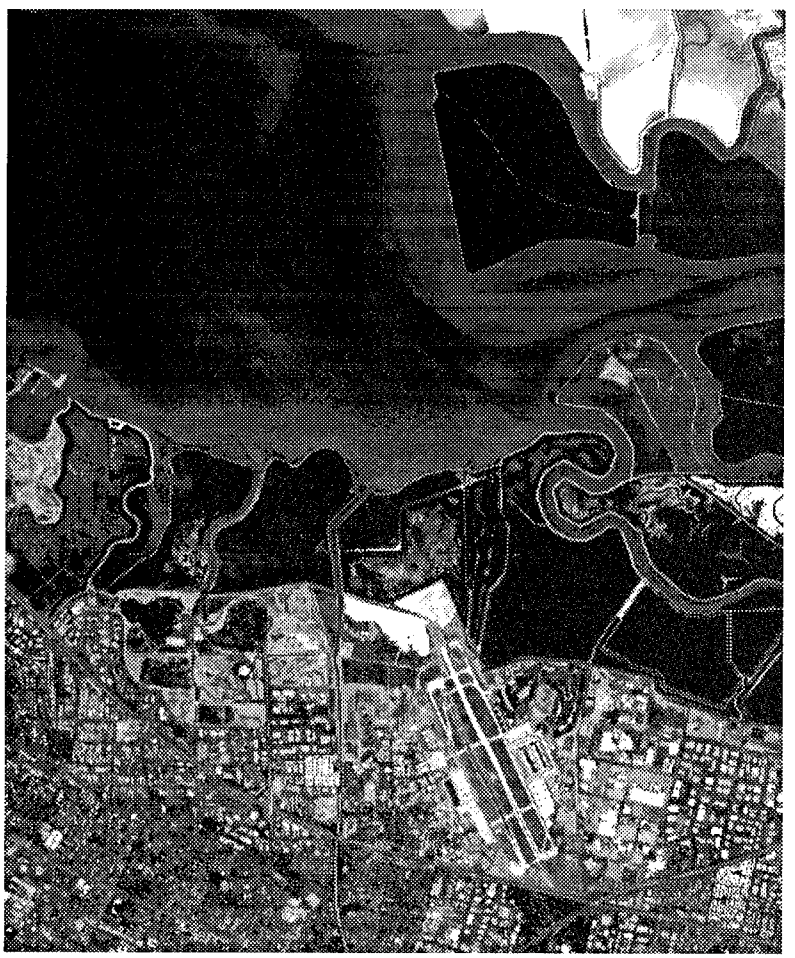


Figure B.4 AVIRIS Hyperspectral Image Representing Band 34

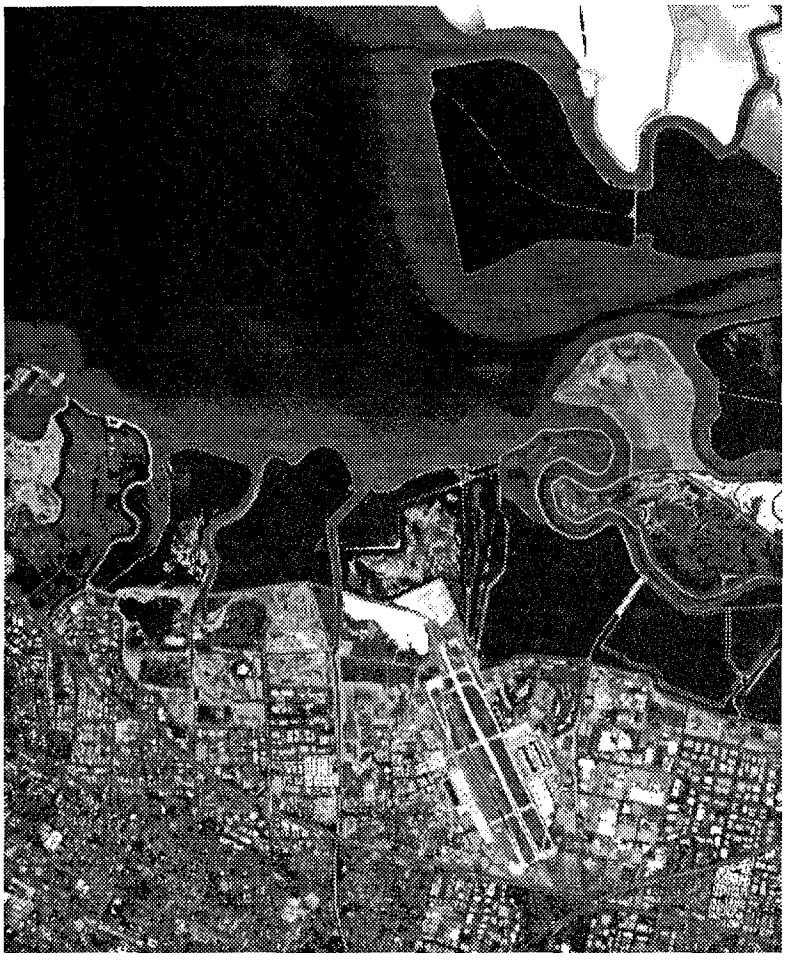


Figure B.5 AVIRIS Hyperspectral Image Representing Band 35

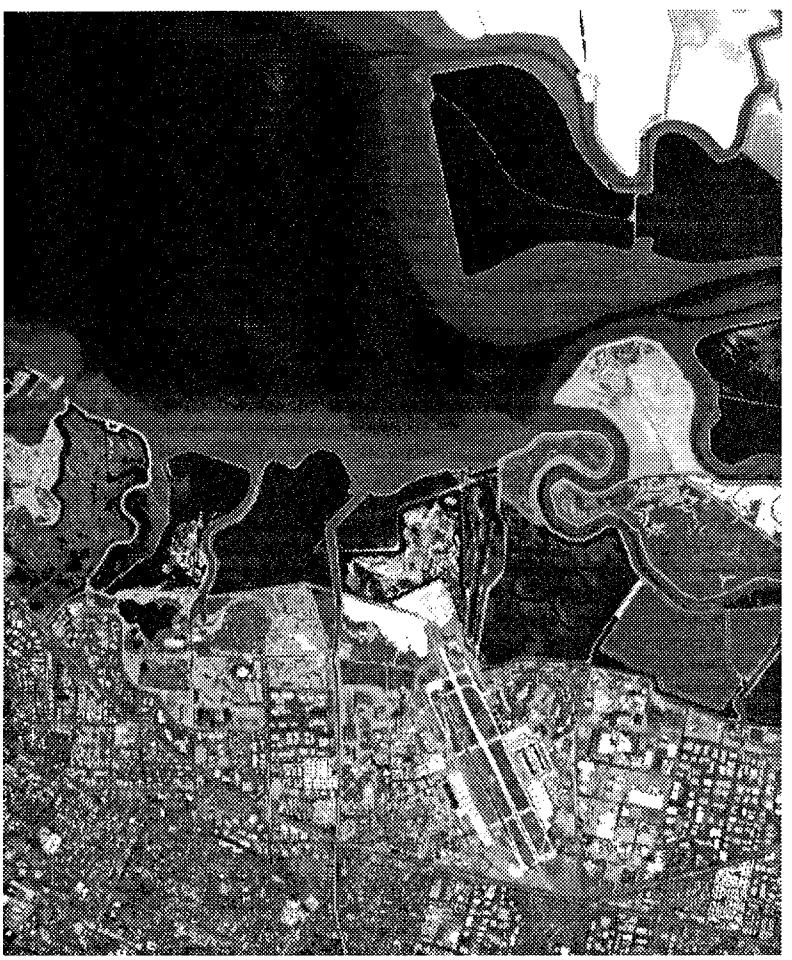


Figure B.6 AVIRIS Hyperspectral Image Representing Band 36

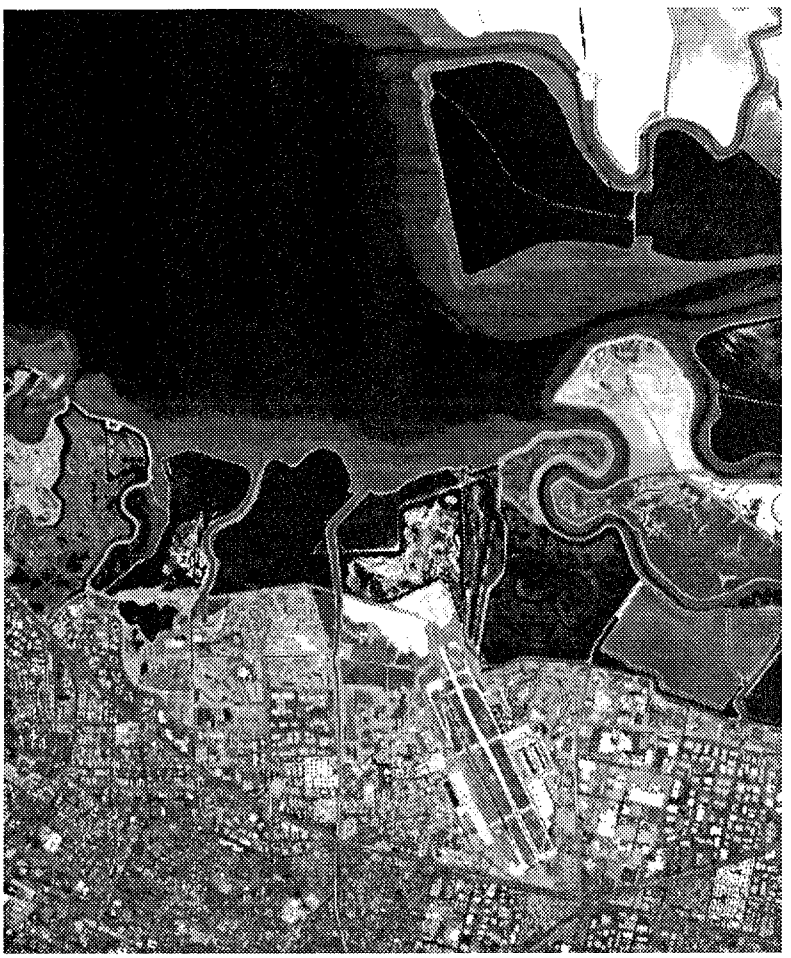


Figure B.7 AVIRIS Hyperspectral Image Representing Band 37

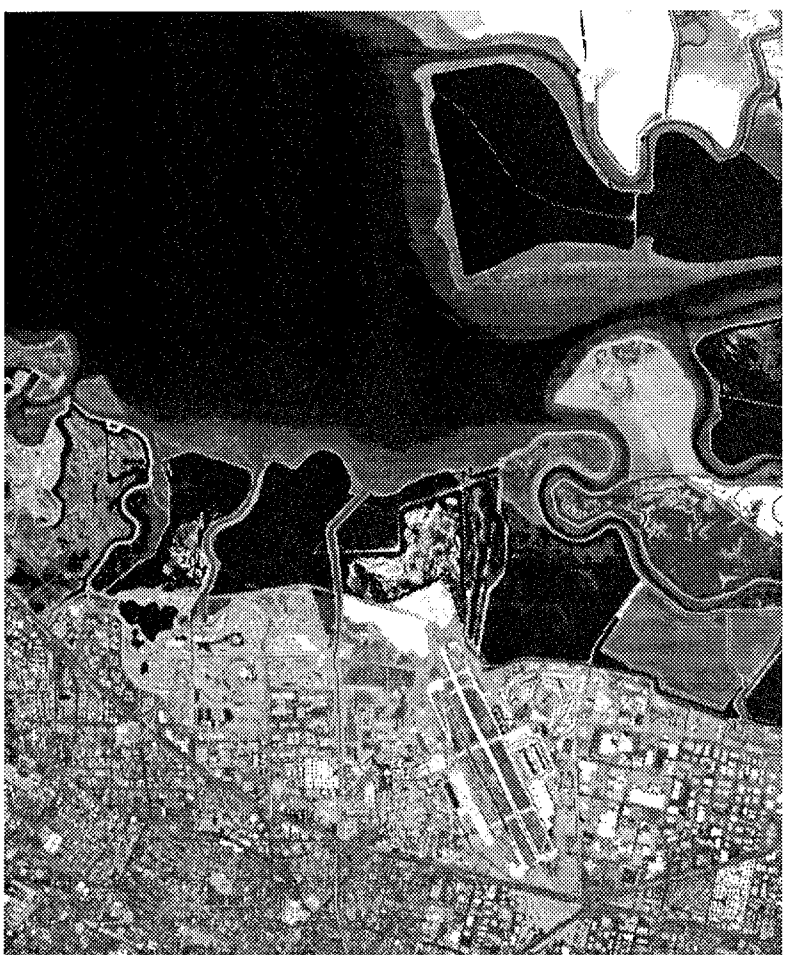


Figure B.8 AVIRIS Hyperspectral Image Representing Band 38

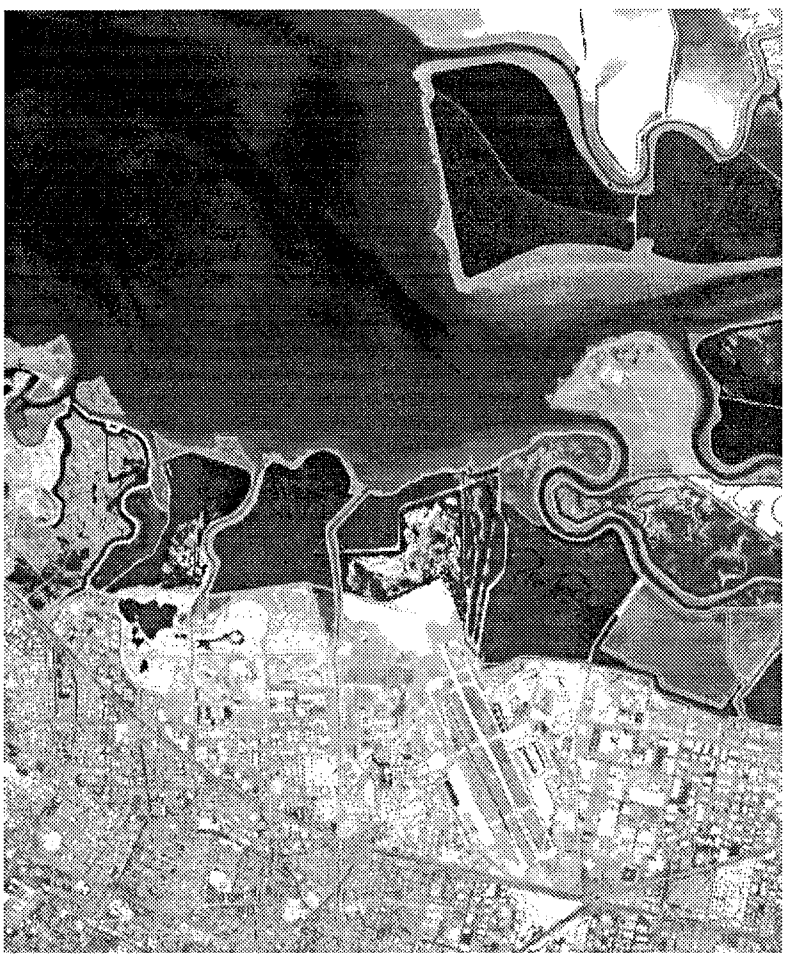


Figure B.9 AVIRIS Hyperspectral Image Representing Band 39

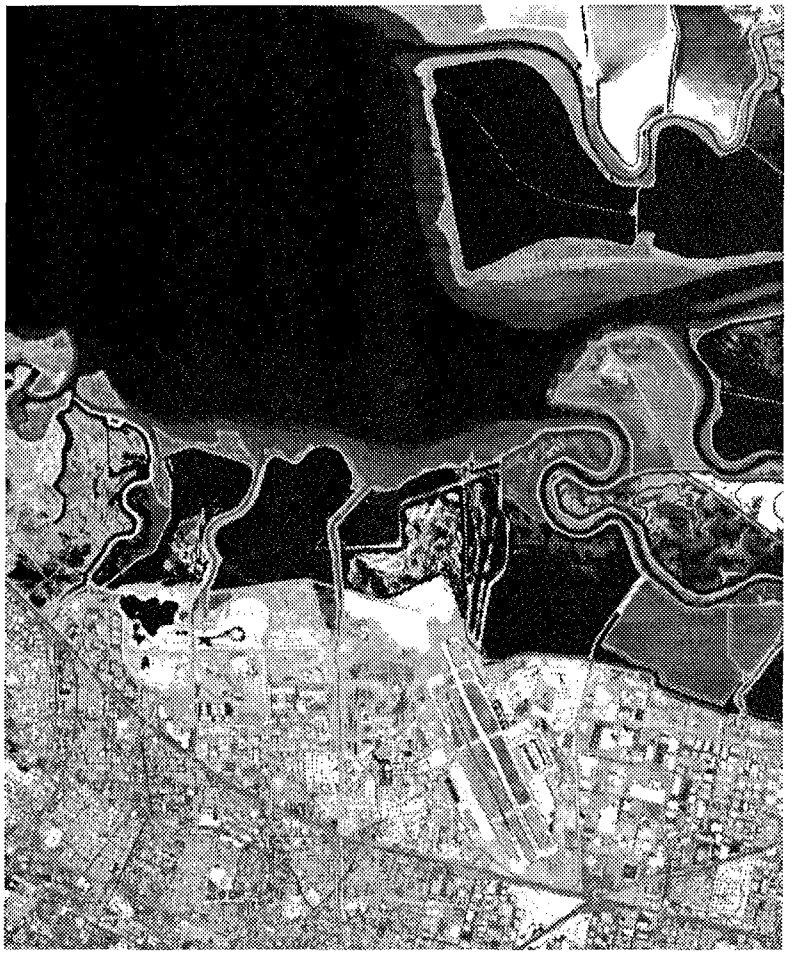


Figure B.10 AVIRIS Hyperspectral Image Representing Band 40

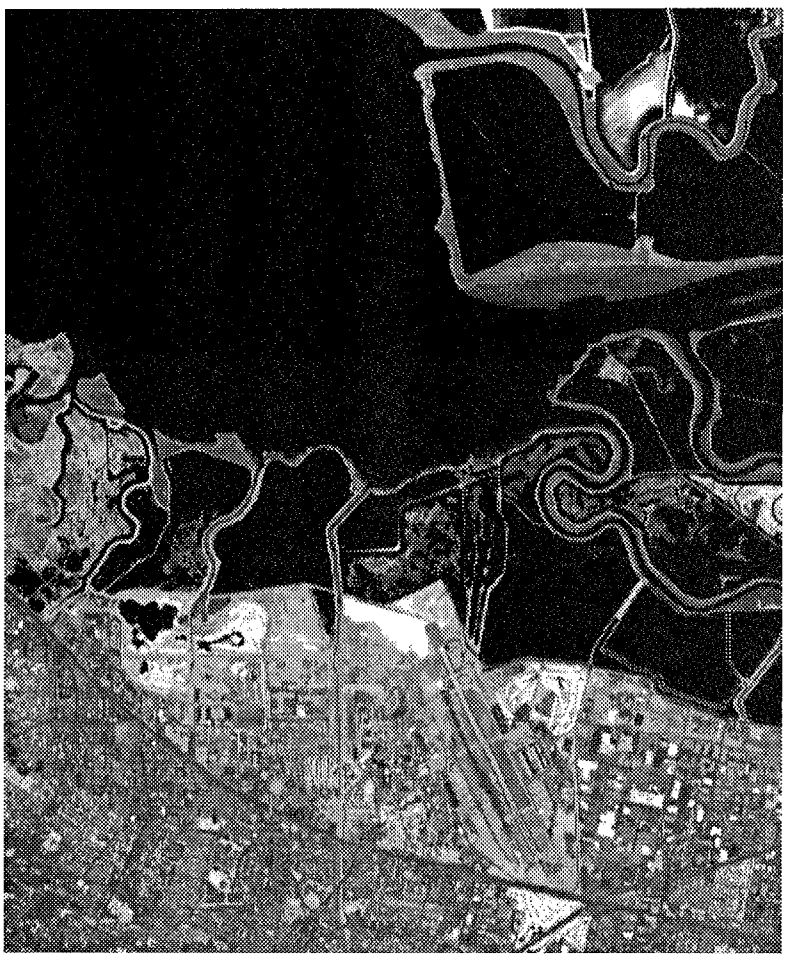


Figure B.11 AVIRIS Hyperspectral Image Representing Band 60

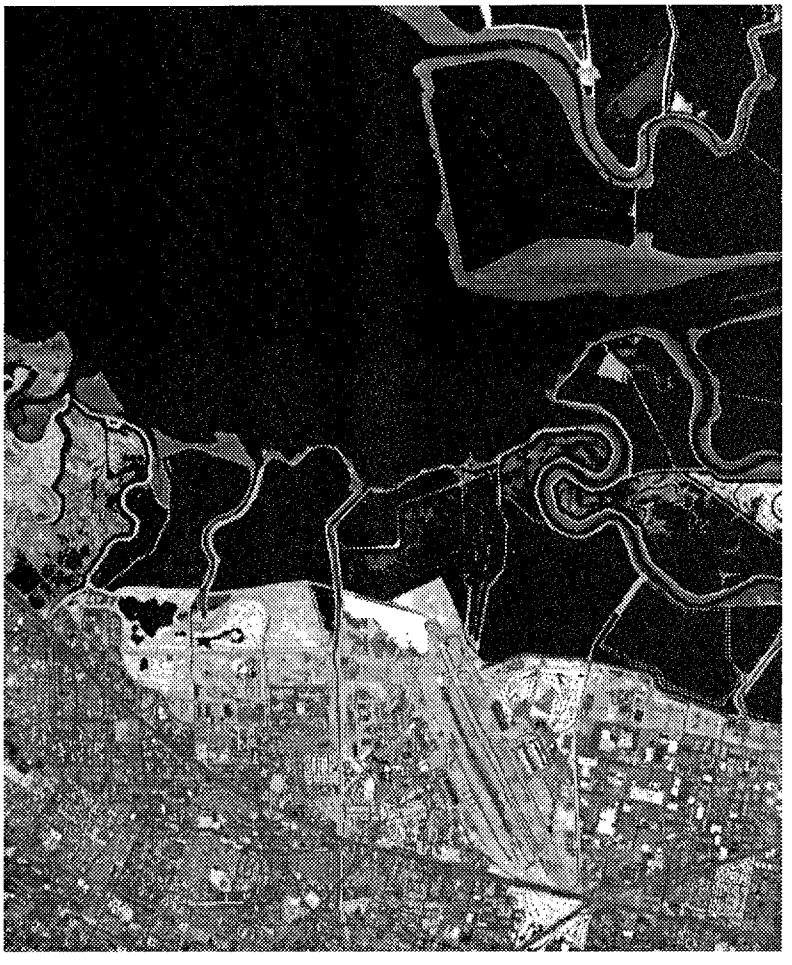


Figure B.12 AVIRIS Hyperspectral Image Representing Band 90

Appendix C. Fusion Results Images

This Appendix provides a figure for every fusion result obtained in this thesis.

C.1 Fusion Results of Test Images

C.2 Synthetic Aperture Radar Image Fusion Results

Figure C.1 Fusion results of test images are arranged as follows: Burt Top left, Toet top right, and Wilson bottom. The input images, which contain uncorrelated noise, are the first three images defined in Table 3.1

Figure C.2 Fusion results of test images are arranged as follows: Burt Top left, Toet top right, and Wilson bottom. The input images, which contain a 5db SNR of uncorrelated noise and are defined in Table 3.1

Figure C.3 Fusion results of test images are arranged as follows: Burt Top left, Toet top right, and Wilson bottom. The input images, which contain a 2.5db SNR of uncorrelated noise and are defined in Table 3.1

Figure C.4 Fusion results of test images are arranged as follows: Burt Top left, Toet top right, and Wilson bottom. The input images, which contain a 5db SNR of correlated noise and are defined in Table 3.1

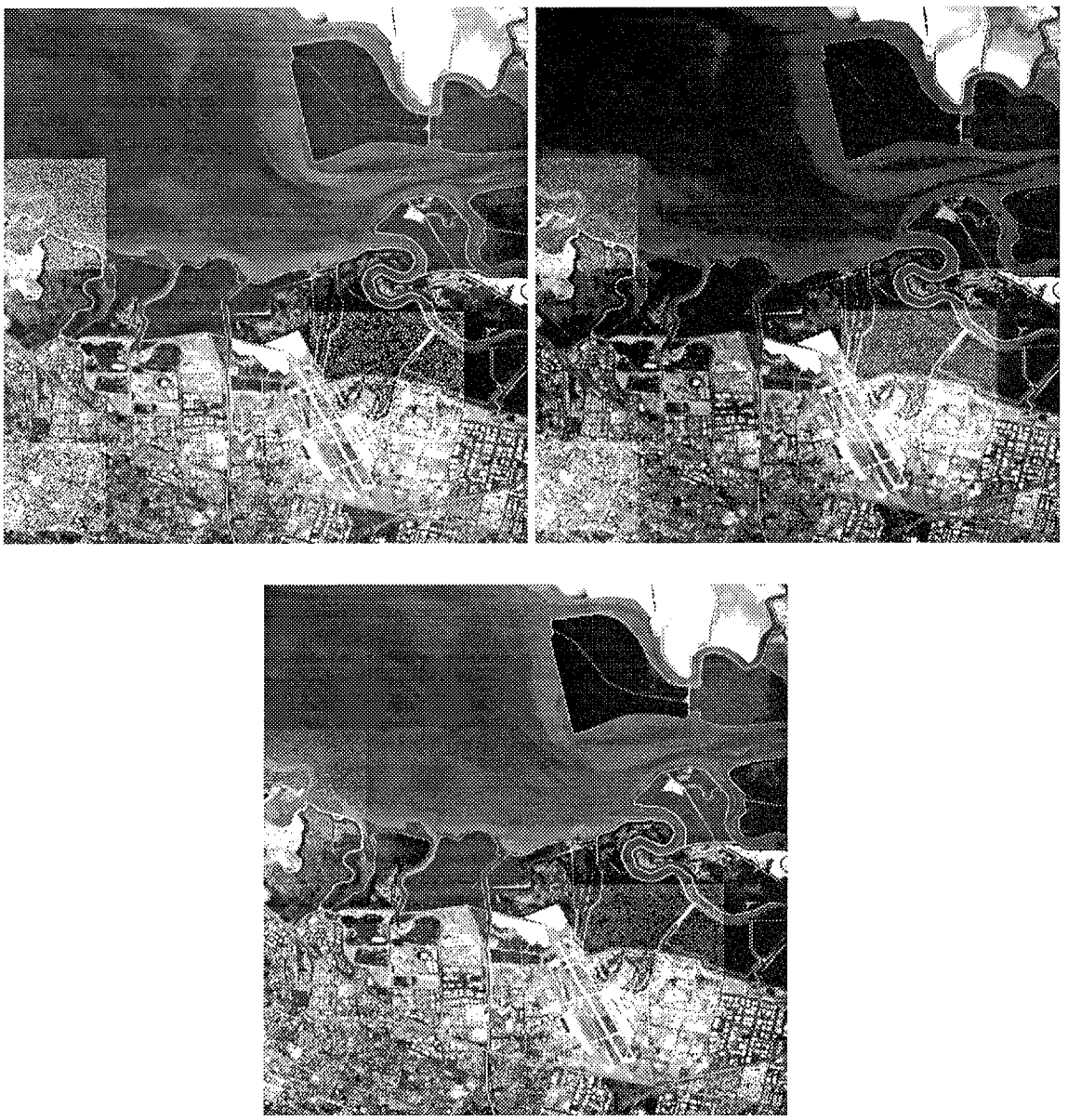


Figure C.5 Fusion results of test images are arranged as follows: Burt Top left, Toet top right, and Wilson bottom. The input images, which contain a 5db SNR of correlated noise and are defined in Table 3.2

C.3 AVIRIS Hyperspectral Image Fusion Results

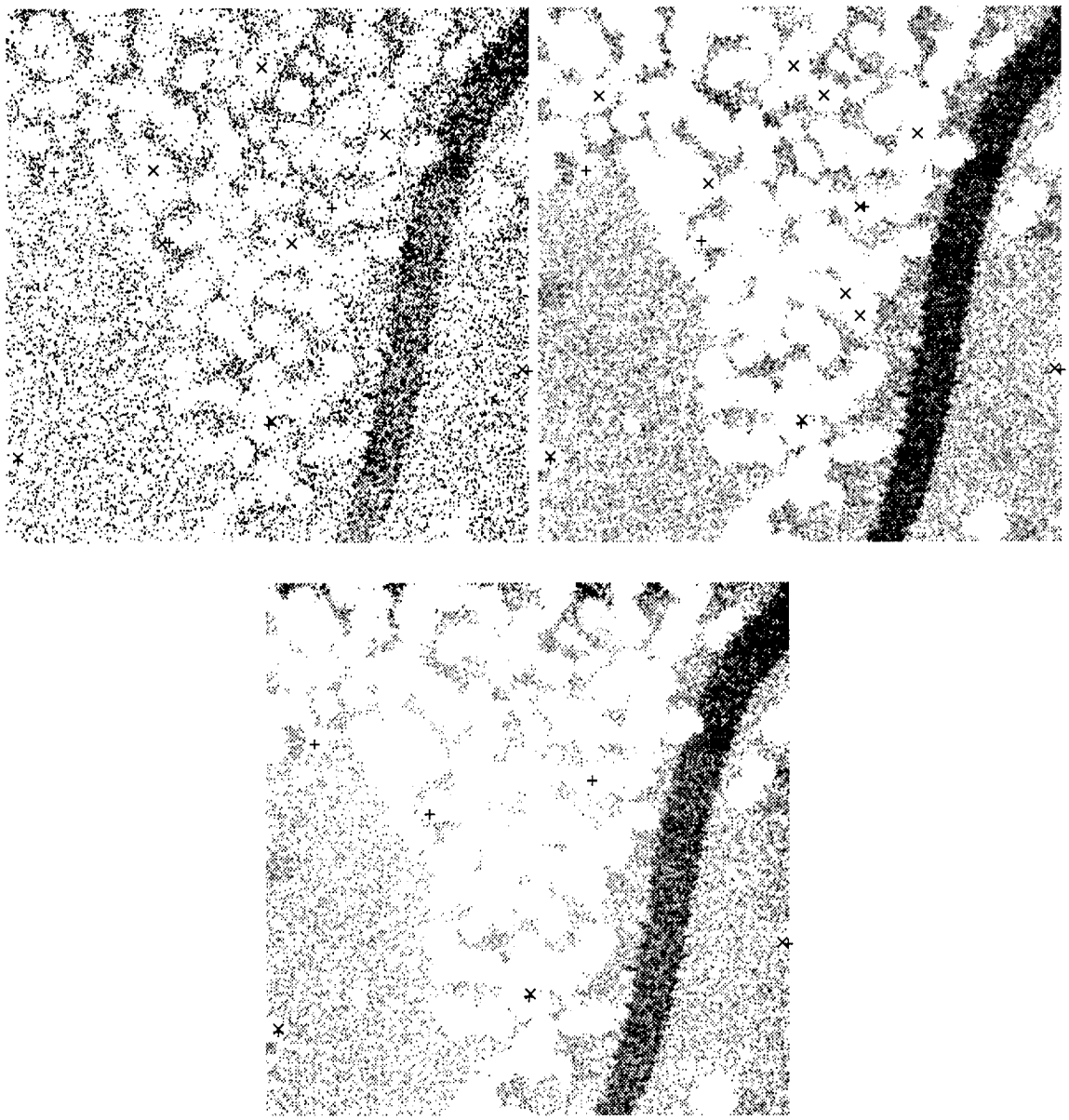


Figure C.6 Fusion results of SAR images are arranged as follows: Burt Top left, Toet top right, and Wilson bottom.

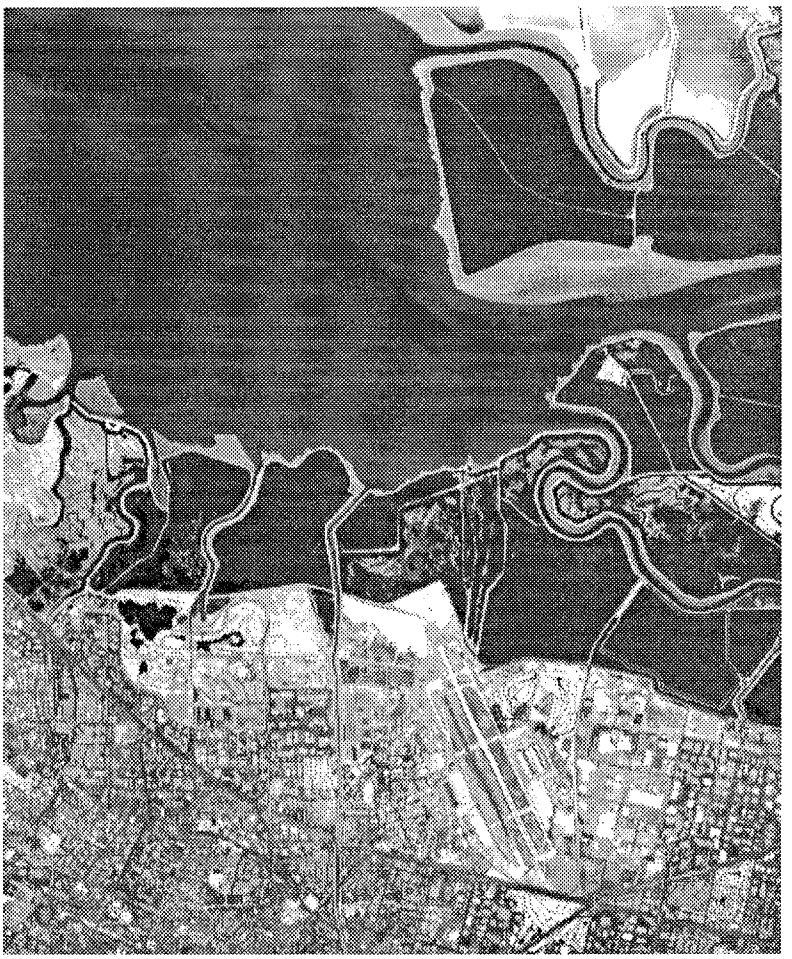


Figure C.7 Burt fusion results of AVIRIS bands 30, 60, and 90.

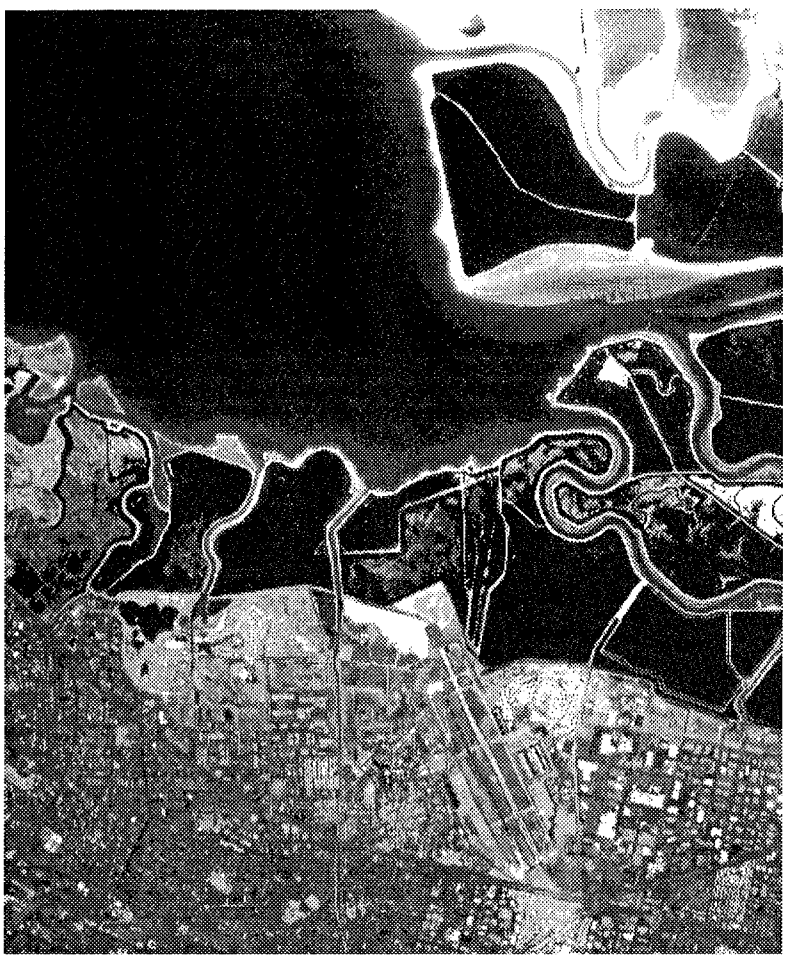


Figure C.8 Toet fusion results of AVIRIS bands 30, 60, and 90.

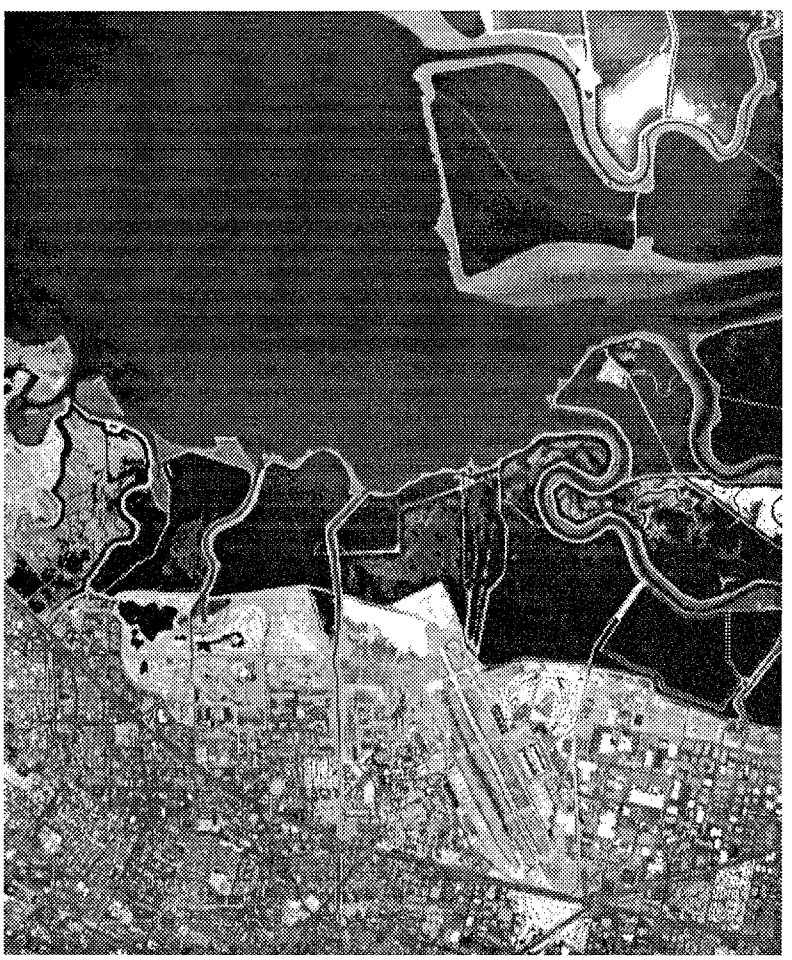


Figure C.9 Wilson fusion results of AVIRIS bands 30, 60, and 90.

Appendix D. Toet Algorithm Implemented with Matlab M-Files

D.1 Image Fusion Algorithm For Bands 30 60 and 90

```
%This program fuses three AVIRIS images.  In this case they are bands 30, 60, 90.  The resulting
% output is a single image that has the same dimensions as the input images, but has now fused all
% three.   Each input  image is padded  before it is deconstructed.  It performs 6 levels of
% deconstruction and the first band (Band 30) is the band that is chosen for the gross approximation.

clear
toetweights

step = 1 % create the filtered and down-sampled pyramid for band30
band30 = getim(30);
band30L1 = reduce(band30,M);
band30L2 = reduce(band30L1,M);
band30L3 = reduce(band30L2,M);
band30L4 = reduce(band30L3,M);
band30L5 = reduce(band30L4,M);
band30L6 = reduce(band30L5,M);

step = 2   %create the ratio pyramid for band30
band30R6 = band30L6;
band30R5 = band30L5 ./ (expand(band30L6,M));
          clear band30L6
band30R4 = band30L4 ./ (expand(band30L5,M));
          clear band30L5
band30R3 = band30L3 ./ (expand(band30L4,M));
          clear band30L4
band30R2 = band30L2 ./ (expand(band30L3,M));
          clear band30L3
band30R1 = band30L1 ./ (expand(band30L2,M));
          clear band30L2
band30R  = band30   ./ (expand(band30L1,M));
          clear band30L1


step = 3 % create the filtered and down-sampled pyramid for band60
band60 = getim(60);
band60L1 = reduce(band60,M);
band60L2 = reduce(band60L1,M);
band60L3 = reduce(band60L2,M);
band60L4 = reduce(band60L3,M);
band60L5 = reduce(band60L4,M);
band60L6 = reduce(band60L5,M);

step = 4 %create the ratio pyramid for band60

band60R6 = band60L6;
band60R5 = band60L5 ./ (expand(band60L6,M));
           clear band60L6
band60R4 = band60L4 ./ (expand(band60L5,M));
           clear band60L5
band60R3 = band60L3 ./ (expand(band60L4,M));
           clear band60L4
band60R2 = band60L2 ./ (expand(band60L3,M));
           clear band60L3
band60R1 = band60L1 ./ (expand(band60L2,M));
           clear band60L2
band60R  = band60   ./ (expand(band60L1,M));
           clear band60L1
```

```
step = 5 % start fusing by creating the max ratio pyramid

diff6 = maxdif(band30R6,band60R6);
clear band30R6
clear band60R6

diff5 = maxdif(band30R5,band60R5);
clear band30R5
clear band60R5

diff4 = maxdif(band30R4,band60R4);
clear band30R4
clear band60R4

diff3 = maxdif(band30R3,band60R3);
clear band30R3
clear band60R3

diff2 = maxdif(band30R2,band60R2);
clear band30R2
clear band60R2

diff1 = maxdif(band30R1,band60R1);
clear band30R1
clear band60R1

diff = maxdif(band30R,band60R);
clear band30R
clear band60R



step = 6 % create the filtered and down-sampled pyramid for band90
band90 = getim(90);
band90L1 = reduce(band90,M);
band90L2 = reduce(band90L1,M);
band90L3 = reduce(band90L2,M);
band90L4 = reduce(band90L3,M);
band90L5 = reduce(band90L4,M);
band90L6 = reduce(band90L5,M);

step = 7   %create the ratio pyramid for band90
band90R6 = band90L6;
band90R5 = band90L5 ./ (expand(band90L6,M));
          clear band90L6
band90R4 = band90L4 ./ (expand(band90L5,M));
          clear band90L5
band90R3 = band90L3 ./ (expand(band90L4,M));
          clear band90L4
band90R2 = band90L2 ./ (expand(band90L3,M));
          clear band90L3
band90R1 = band90L1 ./ (expand(band90L2,M));
          clear band90L2
band90R  = band90   ./ (expand(band90L1,M));
          clear band90L1


step = 8 % start fusing by creating the max ratio pyramid

diff6 = maxdif(band90R6,diff6);
clear band90R6

diff5 = maxdif(band90R5,diff5);
clear band90R5
```

```
diff4 = maxdif(band90R4,diff4);
clear band90R4

diff3 = maxdif(band90R3,diff3);
clear band90R3

diff2 = maxdif(band90R2,diff2);
clear band90R2

diff1 = maxdif(band90R1,diff1);
clear band90R1

diff = maxdif(band90R,diff);
clear band90R


step = 9 %reverse the steps to compress the pyramid to the final image

fuse5 = diff5 .* expand(diff6,M);
clear diff6
clear diff5

fuse4 = diff4 .* expand(fuse5,M);
clear diff4
clear fuse5

fuse3 = diff3 .* expand(fuse4,M);
clear diff3
clear fuse4

fuse2 = diff2 .* expand(fuse3,M);
clear diff2
clear fuse3

fuse1 = diff1 .* expand(fuse2,M);
clear diff1
clear fuse2

G0 = clean(diff .* expand(fuse1,M));
clear diff
clear fuse1

save fu306090 G0
```

D.2 Image Fusion Algorithm For Lenna Test Images

```
 %This program fuses three test images of Lenna.  In this case they are the images 431,432,433
% which relate to the test images of Lenna that have added noise.  The resulting output is a single
% image that has the same dimensions as the input images, but has now fused all three.   Each input
% image is padded  before it is deconstructed.


clear
toetweights

step = 1 % create the filtered and down-sampled pyramid for band431
band431 = getim(431);
band431L1 = reduce(band431,M);
band431L2 = reduce(band431L1,M);
band431L3 = reduce(band431L2,M);
band431L4 = reduce(band431L3,M);
band431L5 = reduce(band431L4,M);
```

```
band431L6 = reduce(band431L5,M);

step = 2   %create the ratio pyramid for band431
band431R6 = band431L6;
band431R5 = band431L5 ./ (expand(band431L6,M));
          clear band431L6
band431R4 = band431L4 ./ (expand(band431L5,M));
          clear band431L5
band431R3 = band431L3 ./ (expand(band431L4,M));
          clear band431L4
band431R2 = band431L2 ./ (expand(band431L3,M));
          clear band431L3
band431R1 = band431L1 ./ (expand(band431L2,M));
          clear band431L2
band431R  = band431   ./ (expand(band431L1,M));
          clear band431L1


step = 3 % create the filtered and down-sampled pyramid for band432
band432 = getim(432);
band432L1 = reduce(band432,M);
band432L2 = reduce(band432L1,M);
band432L3 = reduce(band432L2,M);
band432L4 = reduce(band432L3,M);
band432L5 = reduce(band432L4,M);
band432L6 = reduce(band432L5,M);

step = 4 %create the ratio pyramid for band432

band432R6 = band432L6;
band432R5 = band432L5 ./ (expand(band432L6,M));
           clear band432L6
band432R4 = band432L4 ./ (expand(band432L5,M));
           clear band432L5
band432R3 = band432L3 ./ (expand(band432L4,M));
           clear band432L4
band432R2 = band432L2 ./ (expand(band432L3,M));
           clear band432L3
band432R1 = band432L1 ./ (expand(band432L2,M));
           clear band432L2
band432R  = band432   ./ (expand(band432L1,M));
           clear band432L1

step = 5 % start fusing by creating the max ratio pyramid

diff6 = maxdif(band431R6,band432R6);
clear band431R6
clear band432R6

diff5 = maxdif(band431R5,band432R5);
clear band431R5
clear band432R5

diff4 = maxdif(band431R4,band432R4);
clear band431R4
clear band432R4

diff3 = maxdif(band431R3,band432R3);
clear band431R3
clear band432R3

diff2 = maxdif(band431R2,band432R2);
clear band431R2
clear band432R2
```

```
diff1 = maxdif(band431R1,band432R1);
clear band431R1
clear band432R1

diff = maxdif(band431R,band432R);
clear band431R
clear band432R



step = 6 % create the filtered and down-sampled pyramid for band433
band433 = getim(433);
band433L1 = reduce(band433,M);
band433L2 = reduce(band433L1,M);
band433L3 = reduce(band433L2,M);
band433L4 = reduce(band433L3,M);
band433L5 = reduce(band433L4,M);
band433L6 = reduce(band433L5,M);

step = 7   %create the ratio pyramid for band433
band433R6 = band433L6;
band433R5 = band433L5 ./ (expand(band433L6,M));
          clear band433L6
band433R4 = band433L4 ./ (expand(band433L5,M));
          clear band433L5
band433R3 = band433L3 ./ (expand(band433L4,M));
          clear band433L4
band433R2 = band433L2 ./ (expand(band433L3,M));
          clear band433L3
band433R1 = band433L1 ./ (expand(band433L2,M));
          clear band433L2
band433R  = band433   ./ (expand(band433L1,M));
          clear band433L1


step = 8 % start fusing by creating the max ratio pyramid

diff6 = maxdif(band433R6,diff6);
clear band433R6

diff5 = maxdif(band433R5,diff5);
clear band433R5

diff4 = maxdif(band433R4,diff4);
clear band433R4

diff3 = maxdif(band433R3,diff3);
clear band433R3

diff2 = maxdif(band433R2,diff2);
clear band433R2

diff1 = maxdif(band433R1,diff1);
clear band433R1

diff = maxdif(band433R,diff);
clear band433R


step = 9 %reverse the steps to compress the pyramid to the final image

fuse5 = diff5 .* expand(diff6,M);
clear diff6
clear diff5
```

```
fuse4 = diff4 .* expand(fuse5,M);
clear diff4
clear fuse5

fuse3 = diff3 .* expand(fuse4,M);
clear diff3
clear fuse4

fuse2 = diff2 .* expand(fuse3,M);
clear diff2
clear fuse3

fuse1 = diff1 .* expand(fuse2,M);
clear diff1
clear fuse2

G0 = clean(diff .* expand(fuse1,M));
clear diff
clear fuse1

save tband30noise G0
```

D.3 Image Fusion Algorithm For SAR Imagery

```
%This program fuses three SAR images.  The resulting output is a single image that has the same
% dimensions as the input images, but has now fused all  three.   Each input  image is padded and
% isn't energy normalized before it is deconstructed.  It performs  6 levels of deconstruction and the
% first   SAR image is  chosen for the gross approximation.

clear
toetweights

step = 1 % create the filtered and down-sampled pyramid for band30
band30 = getim_sar('km15hh1');
band30L1 = reduce(band30,M);
band30L2 = reduce(band30L1,M);
band30L3 = reduce(band30L2,M);
band30L4 = reduce(band30L3,M);
band30L5 = reduce(band30L4,M);
band30L6 = reduce(band30L5,M);

step = 2   %create the ratio pyramid for band30
band30R6 = band30L6;
band30R5 = band30L5 ./ (expand(band30L6,M));
          clear band30L6
band30R4 = band30L4 ./ (expand(band30L5,M));
          clear band30L5
band30R3 = band30L3 ./ (expand(band30L4,M));
          clear band30L4
band30R2 = band30L2 ./ (expand(band30L3,M));
          clear band30L3
band30R1 = band30L1 ./ (expand(band30L2,M));
          clear band30L2
band30R  = band30   ./ (expand(band30L1,M));
          clear band30L1


step = 3 % create the filtered and down-sampled pyramid for band60
band60 = getim_sar('km15hv1');
band60L1 = reduce(band60,M);
band60L2 = reduce(band60L1,M);
band60L3 = reduce(band60L2,M);
```

```
band60L4 = reduce(band60L3,M);
band60L5 = reduce(band60L4,M);
band60L6 = reduce(band60L5,M);

step = 4 %create the ratio pyramid for band60

band60R6 = band60L6;
band60R5 = band60L5 ./ (expand(band60L6,M));
           clear band60L6
band60R4 = band60L4 ./ (expand(band60L5,M));
           clear band60L5
band60R3 = band60L3 ./ (expand(band60L4,M));
           clear band60L4
band60R2 = band60L2 ./ (expand(band60L3,M));
           clear band60L3
band60R1 = band60L1 ./ (expand(band60L2,M));
           clear band60L2
band60R  = band60   ./ (expand(band60L1,M));
           clear band60L1

step = 5 % start fusing by creating the max ratio pyramid

diff6 = maxdif(band30R6,band60R6);
clear band30R6
clear band60R6

diff5 = maxdif(band30R5,band60R5);
clear band30R5
clear band60R5

diff4 = maxdif(band30R4,band60R4);
clear band30R4
clear band60R4

diff3 = maxdif(band30R3,band60R3);
clear band30R3
clear band60R3

diff2 = maxdif(band30R2,band60R2);
clear band30R2
clear band60R2

diff1 = maxdif(band30R1,band60R1);
clear band30R1
clear band60R1

diff = maxdif(band30R,band60R);
clear band30R
clear band60R



step = 6 % create the filtered and down-sampled pyramid for band90
band90 = getim_sar('km15vvl');
band90L1 = reduce(band90,M);
band90L2 = reduce(band90L1,M);
band90L3 = reduce(band90L2,M);
band90L4 = reduce(band90L3,M);
band90L5 = reduce(band90L4,M);
band90L6 = reduce(band90L5,M);

step = 7   %create the ratio pyramid for band90
band90R6 = band90L6;
band90R5 = band90L5 ./ (expand(band90L6,M));
```

```
          clear band90L6
band90R4 = band90L4 ./ (expand(band90L5,M));
          clear band90L5
band90R3 = band90L3 ./ (expand(band90L4,M));
          clear band90L4
band90R2 = band90L2 ./ (expand(band90L3,M));
          clear band90L3
band90R1 = band90L1 ./ (expand(band90L2,M));
          clear band90L2
band90R  = band90   ./ (expand(band90L1,M));
          clear band90L1


step = 8 % start fusing by creating the max ratio pyramid

diff6 = maxdif(band90R6,diff6);
clear band90R6

diff5 = maxdif(band90R5,diff5);
clear band90R5

diff4 = maxdif(band90R4,diff4);
clear band90R4

diff3 = maxdif(band90R3,diff3);
clear band90R3

diff2 = maxdif(band90R2,diff2);
clear band90R2

diff1 = maxdif(band90R1,diff1);
clear band90R1

diff = maxdif(band90R,diff);
clear band90R


step = 9 %reverse the steps to compress the pyramid to the final image

fuse5 = diff5 .* expand(diff6,M);
clear diff6
clear diff5

fuse4 = diff4 .* expand(fuse5,M);
clear diff4
clear fuse5

fuse3 = diff3 .* expand(fuse4,M);
clear diff3
clear fuse4

fuse2 = diff2 .* expand(fuse3,M);
clear diff2
clear fuse3

fuse1 = diff1 .* expand(fuse2,M);
clear diff1
clear fuse2

G0 = clean_sar(diff .* expand(fuse1,M));
clear diff
clear fuse1

save fusarm15 G0
```

D.4 Image Fusion Subroutines

D.4.1 Reduction Algorithm.

```
% This function takes in an input matrix  and produces the output layer.  It takes the input layer and
% convolves it with the w weight matrix in toetweights, which is a gaussian type gradient function,
% and then  downsamples by a factor of 2.

function RED = reduce(IMAGE,CONVOLVEFUNCTION)

   TEMP = conv2(IMAGE,CONVOLVEFUNCTION);


 [Y,X] = size(TEMP);

   RED = TEMP((3:2:Y-2),(3:2:X-2));

   clear TEMP  IMAGE CONVOLVEFUNCTION X Y
return
```

D.4.2 Ratio Algorithm.

```
% This function creates the ratio of two input levels from the reduce pyramid.  It does this by
% dividing the lower level (LEVELA) by an expanded upper level (LEVELB).  The output is
% a matrix with the same dimensions as the input level A.

   function RAT = ratio(LEVELA,LEVELB,M)
   temp =  expand(LEVELB,M);

   RAT = LEVELA/temp;

   clear temp
return
```

D.4.3 Maxdif Algorithm.

```
% This function compares the two input matrices point-by-point nad outputs the maximum value of
% each comparison.  The results is a matrix the same size as the input matrices that contains the
% maximum value for each position in the pair wise comparisons.

function MD = maxdif(MATRA,MATRB)
[Y,X] = size(MATRA);
 for i = 1:Y
       for j = 1:X
      MD(i,j) = max([MATRA(i,j) MATRB(i,j)]);
        end
  end
return
```

D.4.4 Weight Matrix Generation Code.

```
%  This is the weight matrix w referred to in Toet's paper on image fusion.
%  It will be used to generate the Gaussian reduced Pyramid.
%
% X = [.4 .5 .2];
  X = [.4 .25 .05];
 for m = -2:2
        for n = -2:2

        M(m+3,n+3) = X(abs(m)+1) * X(abs(n) + 1);
```

```
        end %end for m

      end %end for n

M
```

Appendix E. Burt Matlab M-Files

E.1 Image Fusion Algorithm For Bands 30 60 and 90

```
%This program fuses three AVIRIS images.  In this case they are bands 30, 60, 90.  The resulting
% output is a single image that has the same dimensions as the input images, but has now fused all
% three.   Each input  image is padded and energy normalized before it is deconstructed.  It performs
% 6 levels of deconstruction and the first band (Band 30) is the band that is chosen for the gross
% approximation.  The threshold value is .90.


weights

fuG0 = ennormal(getim(30));

[fuD01 fuD02 fuD03 fuD04] = Orient_Grad_Pyramid(fuG0,w,4);
fuG1 = Grad_reduce(fuG0,W);
clear fuG0

for band = 60:30:90
bandAG0 = ennormal(getim(band));

[bandAD01 bandAD02 bandAD03 bandAD04] = Orient_Grad_Pyramid(bandAG0,w,4);

bandASAL1 = salience(bandAD01);
fuSAL1 = salience(fuD01);
bandmatch = match(bandAD01,fuD01,bandASAL1,fuSAL1);
[Y X] = size(bandmatch);
weightmatrixA = zeros(Y,X);
weightmatrixB = zeros(Y,X);
 alpha = .9
   for i = 1:Y
     for j = 1:X
        if bandmatch(i,j) < alpha
           if bandASAL1(i,j) > fuSAL1(i,j)
              weightmatrixA(i,j) = 1;
           else
              weightmatrixB(i,j) = 1;
           end %if else
        else
           wmin = .5 - .5*((1-bandmatch(i,j))/(1-alpha));
           wmax = 1 - wmin;

           if bandASAL1(i,j) > fuSAL1(i,j)
              weightmatrixA(i,j) = wmax;
              weightmatrixB(i,j) = wmin;
           else
              weightmatrixA(i,j) = wmin;
              weightmatrixB(i,j) = wmax;
           end %end if else

        end %end if else
     end
  end
 fuD01 = (weightmatrixA .* bandAD01) +  (weightmatrixB .* fuD01);

clear weightmatrixA bandAD01 weightmatrixB bandmatch fuSAL1
clear bandASAL1

bandASAL2 = salience(bandAD02);
fuSAL2 = salience(fuD02);
bandmatch = match(bandAD02,fuD02,bandASAL2,fuSAL2);
```

```
[Y X] = size(bandmatch);
weightmatrixA = zeros(Y,X);
weightmatrixB = zeros(Y,X);
 alpha = .9
   for i = 1:Y
     for j = 1:X
        if bandmatch(i,j) < alpha
           if bandASAL2(i,j) > fuSAL2(i,j)
              weightmatrixA(i,j) = 1;
           else
              weightmatrixB(i,j) = 1;
           end
        else
           wmin = .5 - .5*((1-bandmatch(i,j))/(1-alpha));
           wmax = 1 - wmin;

           if bandASAL2(i,j) > fuSAL2(i,j)
              weightmatrixA(i,j) = wmax;
              weightmatrixB(i,j) = wmin;
           else
              weightmatrixA(i,j) = wmin;
              weightmatrixB(i,j) = wmax;
           end

   end
end
end

 fuD02 = (weightmatrixA .* bandAD02) +  (weightmatrixB .* fuD02);

clear weightmatrixA bandAD02 weightmatrixB bandmatch fuSAL2
clear bandASAL2

bandASAL3 = salience(bandAD03);
fuSAL3 = salience(fuD03);
bandmatch = match(bandAD03,fuD03,bandASAL3,fuSAL3);
[Y X] = size(bandmatch);
weightmatrixA = zeros(Y,X);
weightmatrixB = zeros(Y,X);
 alpha = .9
   for i = 1:Y
     for j = 1:X
        if bandmatch(i,j) < alpha
           if bandASAL3(i,j) > fuSAL3(i,j)
              weightmatrixA(i,j) = 1;
           else
              weightmatrixB(i,j) = 1;
           end
        else
           wmin = .5 - .5*((1-bandmatch(i,j))/(1-alpha));
           wmax = 1 - wmin;

           if bandASAL3(i,j) > fuSAL3(i,j)
              weightmatrixA(i,j) = wmax;
              weightmatrixB(i,j) = wmin;
           else
              weightmatrixA(i,j) = wmin;
              weightmatrixB(i,j) = wmax;
           end

   end
end
end

 fuD03 = (weightmatrixA .* bandAD03) +  (weightmatrixB .* fuD03);
```

```
clear weightmatrixA bandAD03 weightmatrixB bandmatch fuSAL3
clear bandASAL3


bandASAL4 = salience(bandAD04);
fuSAL4 = salience(fuD04);
bandmatch = match(bandAD04,fuD04,bandASAL4,fuSAL4);
[Y X] = size(bandmatch);
weightmatrixA = zeros(Y,X);
weightmatrixB = zeros(Y,X);
 alpha = .9
   for i = 1:Y
     for j = 1:X
        if bandmatch(i,j) < alpha
           if bandASAL4(i,j) > fuSAL4(i,j)
             weightmatrixA(i,j) = 1;
           else
             weightmatrixB(i,j) = 1;
           end
        else
           wmin = .5 - .5*((1-bandmatch(i,j))/(1-alpha));
           wmax = 1 - wmin;

           if bandASAL4(i,j) > fuSAL4(i,j)
             weightmatrixA(i,j) = wmax;
             weightmatrixB(i,j) = wmin;
           else
             weightmatrixA(i,j) = wmin;
             weightmatrixB(i,j) = wmax;
           end

   end
end
end

 fuD04 = (weightmatrixA .* bandAD04) +  (weightmatrixB .* fuD04);

clear weightmatrixA bandAD04 weightmatrixB bandmatch fuSAL4
clear bandASAL4

end %for loop

[OLD01 OLD02 OLD03 OLD04] = OLP(fuD01, fuD02, fuD03, fuD04,4);
clear fuD01 fuD02 fuD03 fuD04

[FSD ] = FSDLP(OLD01, OLD02, OLD03, OLD04);
clear OLD01 OLD02 OLD03 OLD04

[RELAP ] = RELP(FSD,W);
clear FSD


[fuD11 fuD12 fuD13 fuD14] = Orient_Grad_Pyramid(fuG1,w,4);
fuG2 = Grad_reduce(fuG1,W);
clear fuG1

for band = 60:30:90,

bandAG0 = ennormal(getim(band));

bandAG1 = Grad_reduce(bandAG0,W);
clear bandAG0

[bandAD11 bandAD12 bandAD13 bandAD14] = Orient_Grad_Pyramid(bandAG1,w,4);
```

```
bandASAL1 = salience(bandAD11);
fuSAL1 = salience(fuD11);
bandmatch = match(bandAD11,fuD11,bandASAL1,fuSAL1);
[Y X] = size(bandmatch);
weightmatrixA = zeros(Y,X);
weightmatrixB = zeros(Y,X);
 alpha = .9
   for i = 1:Y
     for j = 1:X
        if bandmatch(i,j) < alpha
           if bandASAL1(i,j) > fuSAL1(i,j)
             weightmatrixA(i,j) = 1;
           else
             weightmatrixB(i,j) = 1;
           end
        else
           wmin = .5 - .5*((1-bandmatch(i,j))/(1-alpha));
           wmax = 1 - wmin;

           if bandASAL1(i,j) > fuSAL1(i,j)
             weightmatrixA(i,j) = wmax;
             weightmatrixB(i,j) = wmin;
           else
             weightmatrixA(i,j) = wmin;
             weightmatrixB(i,j) = wmax;
           end

   end
end
end

 fuD11 = (weightmatrixA .* bandAD11) +  (weightmatrixB .* fuD11);

clear weightmatrixA bandAD11 weightmatrixB bandmatch fuSAL1
clear bandASAL1

bandASAL2 = salience(bandAD12);
fuSAL2 = salience(fuD12);
bandmatch = match(bandAD12,fuD12,bandASAL2,fuSAL2);
[Y X] = size(bandmatch);
weightmatrixA = zeros(Y,X);
weightmatrixB = zeros(Y,X);
 alpha = .9
   for i = 1:Y
     for j = 1:X
        if bandmatch(i,j) < alpha
           if bandASAL2(i,j) > fuSAL2(i,j)
             weightmatrixA(i,j) = 1;
           else
             weightmatrixB(i,j) = 1;
           end
        else
           wmin = .5 - .5*((1-bandmatch(i,j))/(1-alpha));
           wmax = 1 - wmin;

           if bandASAL2(i,j) > fuSAL2(i,j)
             weightmatrixA(i,j) = wmax;
             weightmatrixB(i,j) = wmin;
           else
             weightmatrixA(i,j) = wmin;
             weightmatrixB(i,j) = wmax;
           end

   end
```

```
end
end

 fuD12 = (weightmatrixA .* bandAD12) +  (weightmatrixB .* fuD12);

clear weightmatrixA bandAD12 weightmatrixB bandmatch fuSAL2
clear bandASAL2

bandASAL3 = salience(bandAD13);
fuSAL3 = salience(fuD13);
bandmatch = match(bandAD13,fuD13,bandASAL3,fuSAL3);
[Y X] = size(bandmatch);
weightmatrixA = zeros(Y,X);
weightmatrixB = zeros(Y,X);
 alpha = .9
   for i = 1:Y
     for j = 1:X
        if bandmatch(i,j) < alpha
           if bandASAL3(i,j) > fuSAL3(i,j)
              weightmatrixA(i,j) = 1;
           else
              weightmatrixB(i,j) = 1;
           end
        else
           wmin = .5 - .5*((1-bandmatch(i,j))/(1-alpha));
           wmax = 1 - wmin;

           if bandASAL3(i,j) > fuSAL3(i,j)
              weightmatrixA(i,j) = wmax;
              weightmatrixB(i,j) = wmin;
           else
              weightmatrixA(i,j) = wmin;
              weightmatrixB(i,j) = wmax;
           end

   end
end
end

 fuD13 = (weightmatrixA .* bandAD13) +  (weightmatrixB .* fuD13);

clear weightmatrixA bandAD13 weightmatrixB bandmatch fuSAL3
clear bandASAL3


bandASAL4 = salience(bandAD14);
fuSAL4 = salience(fuD14);
bandmatch = match(bandAD14,fuD14,bandASAL4,fuSAL4);
[Y X] = size(bandmatch);
weightmatrixA = zeros(Y,X);
weightmatrixB = zeros(Y,X);
 alpha = .9
   for i = 1:Y
     for j = 1:X
        if bandmatch(i,j) < alpha
           if bandASAL4(i,j) > fuSAL4(i,j)
              weightmatrixA(i,j) = 1;
           else
              weightmatrixB(i,j) = 1;
           end
        else
           wmin = .5 - .5*((1-bandmatch(i,j))/(1-alpha));
           wmax = 1 - wmin;

           if bandASAL4(i,j) > fuSAL4(i,j)
```

```
            weightmatrixA(i,j) = wmax;
            weightmatrixB(i,j) = wmin;
          else
            weightmatrixA(i,j) = wmin;
            weightmatrixB(i,j) = wmax;
          end

   end
end
end

 fuD14 = (weightmatrixA .* bandAD14) +  (weightmatrixB .* fuD14);

clear weightmatrixA bandAD14 weightmatrixB bandmatch fuSAL4
clear bandASAL4

end

[OLD11 OLD12 OLD13 OLD14] = OLP(fuD11, fuD12, fuD13, fuD14,4);
clear fuD11 fuD12 fuD13 fuD14

[FSD1] = FSDLP(OLD11, OLD12, OLD13, OLD14);
clear OLD11 OLD12 OLD13 OLD14

[RELAP1] = RELP(FSD1,W);
clear FSD1


[fuD21 fuD22 fuD23 fuD24] = Orient_Grad_Pyramid(fuG2,w,4);
fuG3 = Grad_reduce(fuG2,W);
clear fuG2


for band = 60:30:90,

bandAG0 = ennormal(getim(band));

bandAG1 = Grad_reduce(bandAG0,W);
clear bandAG0
bandAG2 = Grad_reduce(bandAG1,W);
clear bandAG1


[bandAD21 bandAD22 bandAD23 bandAD24] = Orient_Grad_Pyramid(bandAG2,w,4);
bandAG3 = Grad_reduce(bandAG2,W);
clear bandAG2

bandASAL1 = salience(bandAD21);
fuSAL1 = salience(fuD21);
bandmatch = match(bandAD21,fuD21,bandASAL1,fuSAL1);
[Y X] = size(bandmatch);
weightmatrixA = zeros(Y,X);
weightmatrixB = zeros(Y,X);
 alpha = .9
   for i = 1:Y
     for j = 1:X
        if bandmatch(i,j) < alpha
          if bandASAL1(i,j) > fuSAL1(i,j)
            weightmatrixA(i,j) = 1;
          else
            weightmatrixB(i,j) = 1;
          end
        else
          wmin = .5 - .5*((1-bandmatch(i,j))/(1-alpha));
          wmax = 1 - wmin;
```

```
          if bandASAL1(i,j) > fuSAL1(i,j)
            weightmatrixA(i,j) = wmax;
            weightmatrixB(i,j) = wmin;
          else
            weightmatrixA(i,j) = wmin;
            weightmatrixB(i,j) = wmax;
          end

    end
end
end

 fuD21 = (weightmatrixA .* bandAD21) +  (weightmatrixB .* fuD21);

clear weightmatrixA bandAD21 weightmatrixB bandmatch fuSAL1
clear bandASAL1

bandASAL2 = salience(bandAD22);
fuSAL2 = salience(fuD22);
bandmatch = match(bandAD22,fuD22,bandASAL2,fuSAL2);
[Y X] = size(bandmatch);
weightmatrixA = zeros(Y,X);
weightmatrixB = zeros(Y,X);
 alpha = .9
   for i = 1:Y
     for j = 1:X
        if bandmatch(i,j) < alpha
           if bandASAL2(i,j) > fuSAL2(i,j)
             weightmatrixA(i,j) = 1;
           else
             weightmatrixB(i,j) = 1;
           end
        else
           wmin = .5 - .5*((1-bandmatch(i,j))/(1-alpha));
           wmax = 1 - wmin;

           if bandASAL2(i,j) > fuSAL2(i,j)
             weightmatrixA(i,j) = wmax;
             weightmatrixB(i,j) = wmin;
           else
             weightmatrixA(i,j) = wmin;
             weightmatrixB(i,j) = wmax;
           end

    end
end
end

 fuD22 = (weightmatrixA .* bandAD22) +  (weightmatrixB .* fuD22);

clear weightmatrixA bandAD22 weightmatrixB bandmatch fuSAL2
clear bandASAL2

bandASAL3 = salience(bandAD23);
fuSAL3 = salience(fuD23);
bandmatch = match(bandAD23,fuD23,bandASAL3,fuSAL3);
[Y X] = size(bandmatch);
weightmatrixA = zeros(Y,X);
weightmatrixB = zeros(Y,X);
 alpha = .9
   for i = 1:Y
     for j = 1:X
        if bandmatch(i,j) < alpha
           if bandASAL3(i,j) > fuSAL3(i,j)
```

```
            weightmatrixA(i,j) = 1;
          else
            weightmatrixB(i,j) = 1;
          end
        else
          wmin = .5 - .5*((1-bandmatch(i,j))/(1-alpha));
          wmax = 1 - wmin;

          if bandASAL3(i,j) > fuSAL3(i,j)
            weightmatrixA(i,j) = wmax;
            weightmatrixB(i,j) = wmin;
          else
            weightmatrixA(i,j) = wmin;
            weightmatrixB(i,j) = wmax;
          end

    end
end
end

 fuD23 = (weightmatrixA .* bandAD23) +  (weightmatrixB .* fuD23);

clear weightmatrixA bandAD23 weightmatrixB bandmatch fuSAL3
clear bandASAL3


bandASAL4 = salience(bandAD24);
fuSAL4 = salience(fuD24);
bandmatch = match(bandAD24,fuD24,bandASAL4,fuSAL4);
[Y X] = size(bandmatch);
weightmatrixA = zeros(Y,X);
weightmatrixB = zeros(Y,X);
 alpha = .9
   for i = 1:Y
     for j = 1:X
        if bandmatch(i,j) < alpha
          if bandASAL4(i,j) > fuSAL4(i,j)
            weightmatrixA(i,j) = 1;
          else
            weightmatrixB(i,j) = 1;
          end
        else
          wmin = .5 - .5*((1-bandmatch(i,j))/(1-alpha));
          wmax = 1 - wmin;

          if bandASAL4(i,j) > fuSAL4(i,j)
            weightmatrixA(i,j) = wmax;
            weightmatrixB(i,j) = wmin;
          else
            weightmatrixA(i,j) = wmin;
            weightmatrixB(i,j) = wmax;
          end

    end
end
end

 fuD24 = (weightmatrixA .* bandAD24) +  (weightmatrixB .* fuD24);

clear weightmatrixA bandAD24 weightmatrixB bandmatch fuSAL4
clear bandASAL4

end

[OLD21 OLD22 OLD23 OLD24] = OLP(fuD21, fuD22, fuD23, fuD24,4);
```

```
clear fuD21 fuD22 fuD23 fuD24

[FSD2] = FSDLP(OLD21, OLD22, OLD23, OLD24);
clear OLD21 OLD22 OLD23 OLD24

[RELAP2] = RELP(FSD2,W);
clear FSD2

[fuD31 fuD32 fuD33 fuD34] = Orient_Grad_Pyramid(fuG3,w,4);
fuG4 = Grad_reduce(fuG3,W);
clear fuG3


for band = 60:30:90,

bandAG0 = ennormal(getim(band));

bandAG1 = Grad_reduce(bandAG0,W);
clear bandAG0
bandAG2 = Grad_reduce(bandAG1,W);
clear bandAG1
bandAG3 = Grad_reduce(bandAG2,W);
clear bandAG2


[bandAD31 bandAD32 bandAD33 bandAD34] = Orient_Grad_Pyramid(bandAG3,w,4);
clear bandAG3

bandASAL1 = salience(bandAD31);
fuSAL1 = salience(fuD31);
bandmatch = match(bandAD31,fuD31,bandASAL1,fuSAL1);
[Y X] = size(bandmatch);
weightmatrixA = zeros(Y,X);
weightmatrixB = zeros(Y,X);
 alpha = .9
   for i = 1:Y
     for j = 1:X
        if bandmatch(i,j) < alpha
           if bandASAL1(i,j) > fuSAL1(i,j)
              weightmatrixA(i,j) = 1;
           else
              weightmatrixB(i,j) = 1;
           end
        else
           wmin = .5 - .5*((1-bandmatch(i,j))/(1-alpha));
           wmax = 1 - wmin;

           if bandASAL1(i,j) > fuSAL1(i,j)
              weightmatrixA(i,j) = wmax;
              weightmatrixB(i,j) = wmin;
           else
              weightmatrixA(i,j) = wmin;
              weightmatrixB(i,j) = wmax;
           end

   end
end
end

 fuD31 = (weightmatrixA .* bandAD31) +  (weightmatrixB .* fuD31);

clear weightmatrixA bandAD31 weightmatrixB bandmatch fuSAL1
clear bandASAL1

bandASAL2 = salience(bandAD32);
```

```
fuSAL2 = salience(fuD32);
bandmatch = match(bandAD32,fuD32,bandASAL2,fuSAL2);
[Y X] = size(bandmatch);
weightmatrixA = zeros(Y,X);
weightmatrixB = zeros(Y,X);
 alpha = .9
   for i = 1:Y
     for j = 1:X
        if bandmatch(i,j) < alpha
           if bandASAL2(i,j) > fuSAL2(i,j)
              weightmatrixA(i,j) = 1;
           else
              weightmatrixB(i,j) = 1;
           end
        else
           wmin = .5 - .5*((1-bandmatch(i,j))/(1-alpha));
           wmax = 1 - wmin;

           if bandASAL2(i,j) > fuSAL2(i,j)
              weightmatrixA(i,j) = wmax;
              weightmatrixB(i,j) = wmin;
           else
              weightmatrixA(i,j) = wmin;
              weightmatrixB(i,j) = wmax;
           end

   end
end
end

 fuD32 = (weightmatrixA .* bandAD32) +  (weightmatrixB .* fuD32);

clear weightmatrixA bandAD32 weightmatrixB bandmatch fuSAL2
clear bandASAL2

bandASAL3 = salience(bandAD33);
fuSAL3 = salience(fuD33);
bandmatch = match(bandAD33,fuD33,bandASAL3,fuSAL3);
[Y X] = size(bandmatch);
weightmatrixA = zeros(Y,X);
weightmatrixB = zeros(Y,X);
 alpha = .9
   for i = 1:Y
     for j = 1:X
        if bandmatch(i,j) < alpha
           if bandASAL3(i,j) > fuSAL3(i,j)
              weightmatrixA(i,j) = 1;
           else
              weightmatrixB(i,j) = 1;
           end
        else
           wmin = .5 - .5*((1-bandmatch(i,j))/(1-alpha));
           wmax = 1 - wmin;

           if bandASAL3(i,j) > fuSAL3(i,j)
              weightmatrixA(i,j) = wmax;
              weightmatrixB(i,j) = wmin;
           else
              weightmatrixA(i,j) = wmin;
              weightmatrixB(i,j) = wmax;
           end

   end
end
end
```

```
 fuD33 = (weightmatrixA .* bandAD33) +  (weightmatrixB .* fuD33);

clear weightmatrixA bandAD33 weightmatrixB bandmatch fuSAL3
clear bandASAL3


bandASAL4 = salience(bandAD34);
fuSAL4 = salience(fuD34);
bandmatch = match(bandAD34,fuD34,bandASAL4,fuSAL4);
[Y X] = size(bandmatch);
weightmatrixA = zeros(Y,X);
weightmatrixB = zeros(Y,X);
 alpha = .9
   for i = 1:Y
     for j = 1:X
        if bandmatch(i,j) < alpha
           if bandASAL4(i,j) > fuSAL4(i,j)
              weightmatrixA(i,j) = 1;
           else
              weightmatrixB(i,j) = 1;
           end
        else
           wmin = .5 - .5*((1-bandmatch(i,j))/(1-alpha));
           wmax = 1 - wmin;

           if bandASAL4(i,j) > fuSAL4(i,j)
              weightmatrixA(i,j) = wmax;
              weightmatrixB(i,j) = wmin;
           else
              weightmatrixA(i,j) = wmin;
              weightmatrixB(i,j) = wmax;
           end

   end
end
end

 fuD34 = (weightmatrixA .* bandAD34) +  (weightmatrixB .* fuD34);

clear weightmatrixA bandAD34 weightmatrixB bandmatch fuSAL4
clear bandASAL4
end

[OLD31 OLD32 OLD33 OLD34] = OLP(fuD31, fuD32, fuD33, fuD34,4);
clear fuD31 fuD32 fuD33 fuD34

[FSD3] = FSDLP(OLD31, OLD32, OLD33, OLD34);
clear OLD31 OLD32 OLD33 OLD34

[RELAP3] = RELP(FSD3,W);
clear FSD3



[fuD41 fuD42 fuD43 fuD44] = Orient_Grad_Pyramid(fuG4,w,4);
fuG5 = Grad_reduce(fuG4,W);
clear fuG4


for band = 60:30:90
bandAG0 = ennormal(getim(band));

bandAG1 = Grad_reduce(bandAG0,W);
clear bandAG0
```

```
bandAG2 = Grad_reduce(bandAG1,W);
clear bandAG1
bandAG3 = Grad_reduce(bandAG2,W);
clear bandAG2
bandAG4 = Grad_reduce(bandAG3,W);
clear bandAG3

[bandAD41 bandAD42 bandAD43 bandAD44] = Orient_Grad_Pyramid(bandAG4,w,4);
clear bandAG4

bandASAL1 = salience(bandAD41);
fuSAL1 = salience(fuD41);
bandmatch = match(bandAD41,fuD41,bandASAL1,fuSAL1);
[Y X] = size(bandmatch);
weightmatrixA = zeros(Y,X);
weightmatrixB = zeros(Y,X);
 alpha = .9
   for i = 1:Y
     for j = 1:X
        if bandmatch(i,j) < alpha
           if bandASAL1(i,j) > fuSAL1(i,j)
              weightmatrixA(i,j) = 1;
           else
              weightmatrixB(i,j) = 1;
           end %if else
        else
           wmin = .5 - .5*((1-bandmatch(i,j))/(1-alpha));
           wmax = 1 - wmin;

           if bandASAL1(i,j) > fuSAL1(i,j)
              weightmatrixA(i,j) = wmax;
              weightmatrixB(i,j) = wmin;
           else
              weightmatrixA(i,j) = wmin;
              weightmatrixB(i,j) = wmax;
           end %end if else

        end %end if else
     end
  end
 fuD41 = (weightmatrixA .* bandAD41) +  (weightmatrixB .* fuD41);

clear weightmatrixA bandAD41 weightmatrixB bandmatch fuSAL1
clear bandASAL1

bandASAL2 = salience(bandAD42);
fuSAL2 = salience(fuD42);
bandmatch = match(bandAD42,fuD42,bandASAL2,fuSAL2);
[Y X] = size(bandmatch);
weightmatrixA = zeros(Y,X);
weightmatrixB = zeros(Y,X);
 alpha = .9
   for i = 1:Y
     for j = 1:X
        if bandmatch(i,j) < alpha
           if bandASAL2(i,j) > fuSAL2(i,j)
              weightmatrixA(i,j) = 1;
           else
              weightmatrixB(i,j) = 1;
           end
        else
           wmin = .5 - .5*((1-bandmatch(i,j))/(1-alpha));
           wmax = 1 - wmin;

           if bandASAL2(i,j) > fuSAL2(i,j)
```

```
            weightmatrixA(i,j) = wmax;
            weightmatrixB(i,j) = wmin;
          else
            weightmatrixA(i,j) = wmin;
            weightmatrixB(i,j) = wmax;
          end

    end
end
end

 fuD42 = (weightmatrixA .* bandAD42) +  (weightmatrixB .* fuD42);

clear weightmatrixA bandAD42 weightmatrixB bandmatch fuSAL2
clear bandASAL2

bandASAL3 = salience(bandAD43);
fuSAL3 = salience(fuD43);
bandmatch = match(bandAD43,fuD43,bandASAL3,fuSAL3);
[Y X] = size(bandmatch);
weightmatrixA = zeros(Y,X);
weightmatrixB = zeros(Y,X);
 alpha = .9
   for i = 1:Y
     for j = 1:X
        if bandmatch(i,j) < alpha
          if bandASAL3(i,j) > fuSAL3(i,j)
            weightmatrixA(i,j) = 1;
          else
            weightmatrixB(i,j) = 1;
          end
        else
          wmin = .5 - .5*((1-bandmatch(i,j))/(1-alpha));
          wmax = 1 - wmin;

          if bandASAL3(i,j) > fuSAL3(i,j)
            weightmatrixA(i,j) = wmax;
            weightmatrixB(i,j) = wmin;
          else
            weightmatrixA(i,j) = wmin;
            weightmatrixB(i,j) = wmax;
          end

    end
end
end

 fuD43 = (weightmatrixA .* bandAD43) +  (weightmatrixB .* fuD43);

clear weightmatrixA bandAD43 weightmatrixB bandmatch fuSAL3
clear bandASAL3


bandASAL4 = salience(bandAD44);
fuSAL4 = salience(fuD44);
bandmatch = match(bandAD44,fuD44,bandASAL4,fuSAL4);
[Y X] = size(bandmatch);
weightmatrixA = zeros(Y,X);
weightmatrixB = zeros(Y,X);
 alpha = .9
   for i = 1:Y
     for j = 1:X
        if bandmatch(i,j) < alpha
          if bandASAL4(i,j) > fuSAL4(i,j)
            weightmatrixA(i,j) = 1;
```

```
          else
            weightmatrixB(i,j) = 1;
          end
        else
          wmin = .5 - .5*((1-bandmatch(i,j))/(1-alpha));
          wmax = 1 - wmin;

          if bandASAL4(i,j) > fuSAL4(i,j)
            weightmatrixA(i,j) = wmax;
            weightmatrixB(i,j) = wmin;
          else
            weightmatrixA(i,j) = wmin;
            weightmatrixB(i,j) = wmax;
          end

    end
end
end

 fuD44 = (weightmatrixA .* bandAD44) +  (weightmatrixB .* fuD44);

clear weightmatrixA bandAD44 weightmatrixB bandmatch fuSAL4
clear bandASAL4

end %for loop

[OLD41 OLD42 OLD43 OLD44] = OLP(fuD41, fuD42, fuD43, fuD44,4);
clear fuD41 fuD42 fuD43 fuD44

[FSD4 ] = FSDLP(OLD41, OLD42, OLD43, OLD44);
clear OLD41 OLD42 OLD43 OLD44

[RELAP4 ] = RELP(FSD4,W);
clear FSD4



[fuD51 fuD52 fuD53 fuD54] = Orient_Grad_Pyramid(fuG5,w,4);
fuG6 = Grad_reduce(fuG5,W);
clear fuG5


for band = 60:30:90
bandAG0 = ennormal(getim(band));

bandAG1 = Grad_reduce(bandAG0,W);
clear bandAG0
bandAG2 = Grad_reduce(bandAG1,W);
clear bandAG1
bandAG3 = Grad_reduce(bandAG2,W);
clear bandAG2
bandAG4 = Grad_reduce(bandAG3,W);
clear bandAG3
bandAG5 = Grad_reduce(bandAG4,W);
clear bandAG4
[bandAD51 bandAD52 bandAD53 bandAD54] = Orient_Grad_Pyramid(bandAG5,w,4);
clear bandAG5

bandASAL1 = salience(bandAD51);
fuSAL1 = salience(fuD51);
bandmatch = match(bandAD51,fuD51,bandASAL1,fuSAL1);
[Y X] = size(bandmatch);
weightmatrixA = zeros(Y,X);
weightmatrixB = zeros(Y,X);
 alpha = .9
```

```
  for i = 1:Y
    for j = 1:X
      if bandmatch(i,j) < alpha
        if bandASAL1(i,j) > fuSAL1(i,j)
          weightmatrixA(i,j) = 1;
        else
          weightmatrixB(i,j) = 1;
        end %if else
      else
        wmin = .5 - .5*((1-bandmatch(i,j))/(1-alpha));
        wmax = 1 - wmin;

        if bandASAL1(i,j) > fuSAL1(i,j)
          weightmatrixA(i,j) = wmax;
          weightmatrixB(i,j) = wmin;
        else
          weightmatrixA(i,j) = wmin;
          weightmatrixB(i,j) = wmax;
        end %end if else

      end %end if else
    end
  end
 fuD51 = (weightmatrixA .* bandAD51) +  (weightmatrixB .* fuD51);

clear weightmatrixA bandAD51 weightmatrixB bandmatch fuSAL1
clear bandASAL1

bandASAL2 = salience(bandAD52);
fuSAL2 = salience(fuD52);
bandmatch = match(bandAD52,fuD52,bandASAL2,fuSAL2);
[Y X] = size(bandmatch);
weightmatrixA = zeros(Y,X);
weightmatrixB = zeros(Y,X);
 alpha = .9
   for i = 1:Y
     for j = 1:X
       if bandmatch(i,j) < alpha
         if bandASAL2(i,j) > fuSAL2(i,j)
           weightmatrixA(i,j) = 1;
         else
           weightmatrixB(i,j) = 1;
         end
       else
         wmin = .5 - .5*((1-bandmatch(i,j))/(1-alpha));
         wmax = 1 - wmin;

         if bandASAL2(i,j) > fuSAL2(i,j)
           weightmatrixA(i,j) = wmax;
           weightmatrixB(i,j) = wmin;
         else
           weightmatrixA(i,j) = wmin;
           weightmatrixB(i,j) = wmax;
         end

   end
end
end

 fuD52 = (weightmatrixA .* bandAD52) +  (weightmatrixB .* fuD52);

clear weightmatrixA bandAD52 weightmatrixB bandmatch fuSAL2
clear bandASAL2

bandASAL3 = salience(bandAD53);
```

```
fuSAL3 = salience(fuD53);
bandmatch = match(bandAD53,fuD53,bandASAL3,fuSAL3);
[Y X] = size(bandmatch);
weightmatrixA = zeros(Y,X);
weightmatrixB = zeros(Y,X);
 alpha = .9
   for i = 1:Y
     for j = 1:X
        if bandmatch(i,j) < alpha
           if bandASAL3(i,j) > fuSAL3(i,j)
             weightmatrixA(i,j) = 1;
           else
             weightmatrixB(i,j) = 1;
           end
        else
           wmin = .5 - .5*((1-bandmatch(i,j))/(1-alpha));
           wmax = 1 - wmin;

           if bandASAL3(i,j) > fuSAL3(i,j)
             weightmatrixA(i,j) = wmax;
             weightmatrixB(i,j) = wmin;
           else
             weightmatrixA(i,j) = wmin;
             weightmatrixB(i,j) = wmax;
           end

   end
end
end

 fuD53 = (weightmatrixA .* bandAD53) +  (weightmatrixB .* fuD53);

clear weightmatrixA bandAD53 weightmatrixB bandmatch fuSAL3
clear bandASAL3


bandASAL4 = salience(bandAD54);
fuSAL4 = salience(fuD54);
bandmatch = match(bandAD54,fuD54,bandASAL4,fuSAL4);
[Y X] = size(bandmatch);
weightmatrixA = zeros(Y,X);
weightmatrixB = zeros(Y,X);
 alpha = .9
   for i = 1:Y
     for j = 1:X
        if bandmatch(i,j) < alpha
           if bandASAL4(i,j) > fuSAL4(i,j)
             weightmatrixA(i,j) = 1;
           else
             weightmatrixB(i,j) = 1;
           end
        else
           wmin = .5 - .5*((1-bandmatch(i,j))/(1-alpha));
           wmax = 1 - wmin;

           if bandASAL4(i,j) > fuSAL4(i,j)
             weightmatrixA(i,j) = wmax;
             weightmatrixB(i,j) = wmin;
           else
             weightmatrixA(i,j) = wmin;
             weightmatrixB(i,j) = wmax;
           end

   end
end
```

```
end

 fuD54 = (weightmatrixA .* bandAD54) +  (weightmatrixB .* fuD54);

clear weightmatrixA bandAD54 weightmatrixB bandmatch fuSAL4
clear bandASAL4

end %for loop

[OLD51 OLD52 OLD53 OLD54] = OLP(fuD51, fuD52, fuD53, fuD54,4);
clear fuD51 fuD52 fuD53 fuD54

[FSD5 ] = FSDLP(OLD51, OLD52, OLD53, OLD54);
clear OLD51 OLD52 OLD53 OLD54

[RELAP5 ] = RELP(FSD5,W);
clear FSD5

% Now the reconstruction of the Image begins.  Using the results of
% equation (A3) Note: I use the original top level from the Gaussian
% Pyramid to start the reconstruction.  i.e. G6 = bandAG6
band30G0 = ennormal(getim(30));
band30G1 = Grad_reduce(band30G0,W);
clear band30G0
band30G2 = Grad_reduce(band30G1,W);
clear band30G1
band30G3 = Grad_reduce(band30G2,W);
clear band30G2
band30G4 = Grad_reduce(band30G3,W);
clear band30G3
band30G5 = Grad_reduce(band30G4,W);
clear band30G4
G6 = Grad_reduce(band30G5,W);
clear band30G5

G5 = RELAP5 + Gauss_expand(G6,W);
clear RELAP5 G6

G4 = RELAP4 + Gauss_expand(G5,W);
clear RELAP4 G5

G3 = RELAP3 + Gauss_expand(G4,W);
clear RELAP3 G4

G2 = RELAP2 + Gauss_expand(G3,W);
clear RELAP2 G3

G1 = RELAP1 + Gauss_expand(G2,W);
clear RELAP1 G2

G0 = RELAP + Gauss_expand(G1,W);
clear RELAP G1

save   fused3060906 G0

quit
```

E.2 Image Fusion Algorithm For Lenna Test Images

```
%This program fuses three test images of Lenna.  In this case they are the images 331,332,333
% which relate to the test images of Lenna that have added noise.  The resulting output is a single
% image that has the same dimensions as the input images, but has now fused all three.   Each input
% image is padded and energy normalized before it is deconstructed.
```

```
weights    %get the weights used for the decomposition

fuG0 = ennormal(getim512(331));   % energy normalize the input images.

[fuD01 fuD02 fuD03 fuD04] = Orient_Grad_Pyramid(fuG0,w,4);
fuG1 = Grad_reduce(fuG0,W);
clear fuG0

for band = 332:333
bandAG0 = ennormal(getim512(band));

[bandAD01 bandAD02 bandAD03 bandAD04] = Orient_Grad_Pyramid(bandAG0,w,4);

bandASAL1 = salience(bandAD01);
fuSAL1 = salience(fuD01);
bandmatch = match(bandAD01,fuD01,bandASAL1,fuSAL1);
[Y X] = size(bandmatch);
weightmatrixA = zeros(Y,X);
weightmatrixB = zeros(Y,X);
 alpha = .9
   for i = 1:Y
     for j = 1:X
        if bandmatch(i,j) < alpha
           if bandASAL1(i,j) > fuSAL1(i,j)
              weightmatrixA(i,j) = 1;
           else
              weightmatrixB(i,j) = 1;
           end %if else
        else
           wmin = .5 - .5*((1-bandmatch(i,j))/(1-alpha));
           wmax = 1 - wmin;

           if bandASAL1(i,j) > fuSAL1(i,j)
              weightmatrixA(i,j) = wmax;
              weightmatrixB(i,j) = wmin;
           else
              weightmatrixA(i,j) = wmin;
              weightmatrixB(i,j) = wmax;
           end %end if else

        end %end if else
     end
  end
 fuD01 = (weightmatrixA .* bandAD01) +  (weightmatrixB .* fuD01);

clear weightmatrixA bandAD01 weightmatrixB bandmatch fuSAL1
clear bandASAL1

bandASAL2 = salience(bandAD02);
fuSAL2 = salience(fuD02);
bandmatch = match(bandAD02,fuD02,bandASAL2,fuSAL2);
[Y X] = size(bandmatch);
weightmatrixA = zeros(Y,X);
weightmatrixB = zeros(Y,X);
 alpha = .9
   for i = 1:Y
     for j = 1:X
        if bandmatch(i,j) < alpha
           if bandASAL2(i,j) > fuSAL2(i,j)
              weightmatrixA(i,j) = 1;
           else
              weightmatrixB(i,j) = 1;
           end
        else
```

```
          wmin = .5 - .5*((1-bandmatch(i,j))/(1-alpha));
          wmax = 1 - wmin;

          if bandASAL2(i,j) > fuSAL2(i,j)
             weightmatrixA(i,j) = wmax;
             weightmatrixB(i,j) = wmin;
          else
             weightmatrixA(i,j) = wmin;
             weightmatrixB(i,j) = wmax;
          end

   end
end
end

 fuD02 = (weightmatrixA .* bandAD02) +  (weightmatrixB .* fuD02);

clear weightmatrixA bandAD02 weightmatrixB bandmatch fuSAL2
clear bandASAL2

bandASAL3 = salience(bandAD03);
fuSAL3 = salience(fuD03);
bandmatch = match(bandAD03,fuD03,bandASAL3,fuSAL3);
[Y X] = size(bandmatch);
weightmatrixA = zeros(Y,X);
weightmatrixB = zeros(Y,X);
 alpha = .9
   for i = 1:Y
     for j = 1:X
        if bandmatch(i,j) < alpha
           if bandASAL3(i,j) > fuSAL3(i,j)
              weightmatrixA(i,j) = 1;
           else
              weightmatrixB(i,j) = 1;
           end
        else
           wmin = .5 - .5*((1-bandmatch(i,j))/(1-alpha));
           wmax = 1 - wmin;

           if bandASAL3(i,j) > fuSAL3(i,j)
              weightmatrixA(i,j) = wmax;
              weightmatrixB(i,j) = wmin;
           else
              weightmatrixA(i,j) = wmin;
              weightmatrixB(i,j) = wmax;
           end

   end
end
end

 fuD03 = (weightmatrixA .* bandAD03) +  (weightmatrixB .* fuD03);

clear weightmatrixA bandAD03 weightmatrixB bandmatch fuSAL3
clear bandASAL3


bandASAL4 = salience(bandAD04);
fuSAL4 = salience(fuD04);
bandmatch = match(bandAD04,fuD04,bandASAL4,fuSAL4);
[Y X] = size(bandmatch);
weightmatrixA = zeros(Y,X);
weightmatrixB = zeros(Y,X);
 alpha = .9
   for i = 1:Y
```

```
    for j = 1:X
      if bandmatch(i,j) < alpha
        if bandASAL4(i,j) > fuSAL4(i,j)
          weightmatrixA(i,j) = 1;
        else
          weightmatrixB(i,j) = 1;
        end
      else
        wmin = .5 - .5*((1-bandmatch(i,j))/(1-alpha));
        wmax = 1 - wmin;

        if bandASAL4(i,j) > fuSAL4(i,j)
          weightmatrixA(i,j) = wmax;
          weightmatrixB(i,j) = wmin;
        else
          weightmatrixA(i,j) = wmin;
          weightmatrixB(i,j) = wmax;
        end

  end
end
end

 fuD04 = (weightmatrixA .* bandAD04) +  (weightmatrixB .* fuD04);

clear weightmatrixA bandAD04 weightmatrixB bandmatch fuSAL4
clear bandASAL4

end %for loop

[OLD01 OLD02 OLD03 OLD04] = OLP(fuD01, fuD02, fuD03, fuD04,4);
clear fuD01 fuD02 fuD03 fuD04

[FSD ] = FSDLP(OLD01, OLD02, OLD03, OLD04);
clear OLD01 OLD02 OLD03 OLD04

[RELAP ] = RELP(FSD,W);
clear FSD


[fuD11 fuD12 fuD13 fuD14] = Orient_Grad_Pyramid(fuG1,w,4);
fuG2 = Grad_reduce(fuG1,W);
clear fuG1

for band = 332:333,

bandAG0 = ennormal(getim512(band));

bandAG1 = Grad_reduce(bandAG0,W);
clear bandAG0

[bandAD11 bandAD12 bandAD13 bandAD14] = Orient_Grad_Pyramid(bandAG1,w,4);

bandASAL1 = salience(bandAD11);
fuSAL1 = salience(fuD11);
bandmatch = match(bandAD11,fuD11,bandASAL1,fuSAL1);
[Y X] = size(bandmatch);
weightmatrixA = zeros(Y,X);
weightmatrixB = zeros(Y,X);
 alpha = .9
   for i = 1:Y
     for j = 1:X
       if bandmatch(i,j) < alpha
         if bandASAL1(i,j) > fuSAL1(i,j)
           weightmatrixA(i,j) = 1;
```

```
          else
            weightmatrixB(i,j) = 1;
          end
        else
          wmin = .5 - .5*((1-bandmatch(i,j))/(1-alpha));
          wmax = 1 - wmin;

          if bandASAL1(i,j) > fuSAL1(i,j)
            weightmatrixA(i,j) = wmax;
            weightmatrixB(i,j) = wmin;
          else
            weightmatrixA(i,j) = wmin;
            weightmatrixB(i,j) = wmax;
          end

    end
end
end

 fuD11 = (weightmatrixA .* bandAD11) +  (weightmatrixB .* fuD11);

clear weightmatrixA bandAD11 weightmatrixB bandmatch fuSAL1
clear bandASAL1

bandASAL2 = salience(bandAD12);
fuSAL2 = salience(fuD12);
bandmatch = match(bandAD12,fuD12,bandASAL2,fuSAL2);
[Y X] = size(bandmatch);
weightmatrixA = zeros(Y,X);
weightmatrixB = zeros(Y,X);
 alpha = .9
   for i = 1:Y
     for j = 1:X
        if bandmatch(i,j) < alpha
          if bandASAL2(i,j) > fuSAL2(i,j)
            weightmatrixA(i,j) = 1;
          else
            weightmatrixB(i,j) = 1;
          end
        else
          wmin = .5 - .5*((1-bandmatch(i,j))/(1-alpha));
          wmax = 1 - wmin;

          if bandASAL2(i,j) > fuSAL2(i,j)
            weightmatrixA(i,j) = wmax;
            weightmatrixB(i,j) = wmin;
          else
            weightmatrixA(i,j) = wmin;
            weightmatrixB(i,j) = wmax;
          end

    end
end
end

 fuD12 = (weightmatrixA .* bandAD12) +  (weightmatrixB .* fuD12);

clear weightmatrixA bandAD12 weightmatrixB bandmatch fuSAL2
clear bandASAL2

bandASAL3 = salience(bandAD13);
fuSAL3 = salience(fuD13);
bandmatch = match(bandAD13,fuD13,bandASAL3,fuSAL3);
[Y X] = size(bandmatch);
weightmatrixA = zeros(Y,X);
```

```
weightmatrixB = zeros(Y,X);
 alpha = .9
   for i = 1:Y
     for j = 1:X
        if bandmatch(i,j) < alpha
           if bandASAL3(i,j) > fuSAL3(i,j)
              weightmatrixA(i,j) = 1;
           else
              weightmatrixB(i,j) = 1;
           end
        else
           wmin = .5 - .5*((1-bandmatch(i,j))/(1-alpha));
           wmax = 1 - wmin;

           if bandASAL3(i,j) > fuSAL3(i,j)
              weightmatrixA(i,j) = wmax;
              weightmatrixB(i,j) = wmin;
           else
              weightmatrixA(i,j) = wmin;
              weightmatrixB(i,j) = wmax;
           end

   end
end
end

 fuD13 = (weightmatrixA .* bandAD13) +  (weightmatrixB .* fuD13);

clear weightmatrixA bandAD13 weightmatrixB bandmatch fuSAL3
clear bandASAL3


bandASAL4 = salience(bandAD14);
fuSAL4 = salience(fuD14);
bandmatch = match(bandAD14,fuD14,bandASAL4,fuSAL4);
[Y X] = size(bandmatch);
weightmatrixA = zeros(Y,X);
weightmatrixB = zeros(Y,X);
 alpha = .9
   for i = 1:Y
     for j = 1:X
        if bandmatch(i,j) < alpha
           if bandASAL4(i,j) > fuSAL4(i,j)
              weightmatrixA(i,j) = 1;
           else
              weightmatrixB(i,j) = 1;
           end
        else
           wmin = .5 - .5*((1-bandmatch(i,j))/(1-alpha));
           wmax = 1 - wmin;

           if bandASAL4(i,j) > fuSAL4(i,j)
              weightmatrixA(i,j) = wmax;
              weightmatrixB(i,j) = wmin;
           else
              weightmatrixA(i,j) = wmin;
              weightmatrixB(i,j) = wmax;
           end

   end
end
end

 fuD14 = (weightmatrixA .* bandAD14) +  (weightmatrixB .* fuD14);
```

```
clear weightmatrixA bandAD14 weightmatrixB bandmatch fuSAL4
clear bandASAL4

end

[OLD11 OLD12 OLD13 OLD14] = OLP(fuD11, fuD12, fuD13, fuD14,4);
clear fuD11 fuD12 fuD13 fuD14

[FSD1] = FSDLP(OLD11, OLD12, OLD13, OLD14);
clear OLD11 OLD12 OLD13 OLD14

[RELAP1] = RELP(FSD1,W);
clear FSD1


[fuD21 fuD22 fuD23 fuD24] = Orient_Grad_Pyramid(fuG2,w,4);
fuG3 = Grad_reduce(fuG2,W);
clear fuG2


for band = 332:333,

bandAG0 = ennormal(getim512(band));

bandAG1 = Grad_reduce(bandAG0,W);
clear bandAG0
bandAG2 = Grad_reduce(bandAG1,W);
clear bandAG1


[bandAD21 bandAD22 bandAD23 bandAD24] = Orient_Grad_Pyramid(bandAG2,w,4);
bandAG3 = Grad_reduce(bandAG2,W);
clear bandAG2

bandASAL1 = salience(bandAD21);
fuSAL1 = salience(fuD21);
bandmatch = match(bandAD21,fuD21,bandASAL1,fuSAL1);
[Y X] = size(bandmatch);
weightmatrixA = zeros(Y,X);
weightmatrixB = zeros(Y,X);
 alpha = .9
   for i = 1:Y
     for j = 1:X
        if bandmatch(i,j) < alpha
           if bandASAL1(i,j) > fuSAL1(i,j)
              weightmatrixA(i,j) = 1;
           else
              weightmatrixB(i,j) = 1;
           end
        else
           wmin = .5 - .5*((1-bandmatch(i,j))/(1-alpha));
           wmax = 1 - wmin;

           if bandASAL1(i,j) > fuSAL1(i,j)
              weightmatrixA(i,j) = wmax;
              weightmatrixB(i,j) = wmin;
           else
              weightmatrixA(i,j) = wmin;
              weightmatrixB(i,j) = wmax;
           end

   end
end
end
```

```
 fuD21 = (weightmatrixA .* bandAD21) +  (weightmatrixB .* fuD21);

clear weightmatrixA bandAD21 weightmatrixB bandmatch fuSAL1
clear bandASAL1

bandASAL2 = salience(bandAD22);
fuSAL2 = salience(fuD22);
bandmatch = match(bandAD22,fuD22,bandASAL2,fuSAL2);
[Y X] = size(bandmatch);
weightmatrixA = zeros(Y,X);
weightmatrixB = zeros(Y,X);
 alpha = .9
   for i = 1:Y
     for j = 1:X
        if bandmatch(i,j) < alpha
           if bandASAL2(i,j) > fuSAL2(i,j)
              weightmatrixA(i,j) = 1;
           else
              weightmatrixB(i,j) = 1;
           end
        else
           wmin = .5 - .5*((1-bandmatch(i,j))/(1-alpha));
           wmax = 1 - wmin;

           if bandASAL2(i,j) > fuSAL2(i,j)
              weightmatrixA(i,j) = wmax;
              weightmatrixB(i,j) = wmin;
           else
              weightmatrixA(i,j) = wmin;
              weightmatrixB(i,j) = wmax;
           end

   end
end
end

 fuD22 = (weightmatrixA .* bandAD22) +  (weightmatrixB .* fuD22);

clear weightmatrixA bandAD22 weightmatrixB bandmatch fuSAL2
clear bandASAL2

bandASAL3 = salience(bandAD23);
fuSAL3 = salience(fuD23);
bandmatch = match(bandAD23,fuD23,bandASAL3,fuSAL3);
[Y X] = size(bandmatch);
weightmatrixA = zeros(Y,X);
weightmatrixB = zeros(Y,X);
 alpha = .9
   for i = 1:Y
     for j = 1:X
        if bandmatch(i,j) < alpha
           if bandASAL3(i,j) > fuSAL3(i,j)
              weightmatrixA(i,j) = 1;
           else
              weightmatrixB(i,j) = 1;
           end
        else
           wmin = .5 - .5*((1-bandmatch(i,j))/(1-alpha));
           wmax = 1 - wmin;

           if bandASAL3(i,j) > fuSAL3(i,j)
              weightmatrixA(i,j) = wmax;
              weightmatrixB(i,j) = wmin;
           else
              weightmatrixA(i,j) = wmin;
```

```
            weightmatrixB(i,j) = wmax;
          end

    end
end
end

 fuD23 = (weightmatrixA .* bandAD23) +  (weightmatrixB .* fuD23);

clear weightmatrixA bandAD23 weightmatrixB bandmatch fuSAL3
clear bandASAL3


bandASAL4 = salience(bandAD24);
fuSAL4 = salience(fuD24);
bandmatch = match(bandAD24,fuD24,bandASAL4,fuSAL4);
[Y X] = size(bandmatch);
weightmatrixA = zeros(Y,X);
weightmatrixB = zeros(Y,X);
 alpha = .9
   for i = 1:Y
     for j = 1:X
        if bandmatch(i,j) < alpha
          if bandASAL4(i,j) > fuSAL4(i,j)
            weightmatrixA(i,j) = 1;
          else
            weightmatrixB(i,j) = 1;
          end
        else
          wmin = .5 - .5*((1-bandmatch(i,j))/(1-alpha));
          wmax = 1 - wmin;

          if bandASAL4(i,j) > fuSAL4(i,j)
            weightmatrixA(i,j) = wmax;
            weightmatrixB(i,j) = wmin;
          else
            weightmatrixA(i,j) = wmin;
            weightmatrixB(i,j) = wmax;
          end

    end
end
end

 fuD24 = (weightmatrixA .* bandAD24) +  (weightmatrixB .* fuD24);

clear weightmatrixA bandAD24 weightmatrixB bandmatch fuSAL4
clear bandASAL4

end

[OLD21 OLD22 OLD23 OLD24] = OLP(fuD21, fuD22, fuD23, fuD24,4);
clear fuD21 fuD22 fuD23 fuD24

[FSD2] = FSDLP(OLD21, OLD22, OLD23, OLD24);
clear OLD21 OLD22 OLD23 OLD24

[RELAP2] = RELP(FSD2,W);
clear FSD2

[fuD31 fuD32 fuD33 fuD34] = Orient_Grad_Pyramid(fuG3,w,4);
fuG4 = Grad_reduce(fuG3,W);
clear fuG3
```

```
for band = 332:333,

bandAG0 = ennormal(getim512(band));

bandAG1 = Grad_reduce(bandAG0,W);
clear bandAG0
bandAG2 = Grad_reduce(bandAG1,W);
clear bandAG1
bandAG3 = Grad_reduce(bandAG2,W);
clear bandAG2


[bandAD31 bandAD32 bandAD33 bandAD34] = Orient_Grad_Pyramid(bandAG3,w,4);
clear bandAG3

bandASAL1 = salience(bandAD31);
fuSAL1 = salience(fuD31);
bandmatch = match(bandAD31,fuD31,bandASAL1,fuSAL1);
[Y X] = size(bandmatch);
weightmatrixA = zeros(Y,X);
weightmatrixB = zeros(Y,X);
 alpha = .9
   for i = 1:Y
     for j = 1:X
        if bandmatch(i,j) < alpha
           if bandASAL1(i,j) > fuSAL1(i,j)
             weightmatrixA(i,j) = 1;
           else
             weightmatrixB(i,j) = 1;
           end
        else
           wmin = .5 - .5*((1-bandmatch(i,j))/(1-alpha));
           wmax = 1 - wmin;

           if bandASAL1(i,j) > fuSAL1(i,j)
             weightmatrixA(i,j) = wmax;
             weightmatrixB(i,j) = wmin;
           else
             weightmatrixA(i,j) = wmin;
             weightmatrixB(i,j) = wmax;
           end

   end
end
end

 fuD31 = (weightmatrixA .* bandAD31) +  (weightmatrixB .* fuD31);

clear weightmatrixA bandAD31 weightmatrixB bandmatch fuSAL1
clear bandASAL1

bandASAL2 = salience(bandAD32);
fuSAL2 = salience(fuD32);
bandmatch = match(bandAD32,fuD32,bandASAL2,fuSAL2);
[Y X] = size(bandmatch);
weightmatrixA = zeros(Y,X);
weightmatrixB = zeros(Y,X);
 alpha = .9
   for i = 1:Y
     for j = 1:X
        if bandmatch(i,j) < alpha
           if bandASAL2(i,j) > fuSAL2(i,j)
             weightmatrixA(i,j) = 1;
           else
             weightmatrixB(i,j) = 1;
```

```
        end
      else
        wmin = .5 - .5*((1-bandmatch(i,j))/(1-alpha));
        wmax = 1 - wmin;

        if bandASAL2(i,j) > fuSAL2(i,j)
          weightmatrixA(i,j) = wmax;
          weightmatrixB(i,j) = wmin;
        else
          weightmatrixA(i,j) = wmin;
          weightmatrixB(i,j) = wmax;
        end

  end
end
end

 fuD32 = (weightmatrixA .* bandAD32) +  (weightmatrixB .* fuD32);

clear weightmatrixA bandAD32 weightmatrixB bandmatch fuSAL2
clear bandASAL2

bandASAL3 = salience(bandAD33);
fuSAL3 = salience(fuD33);
bandmatch = match(bandAD33,fuD33,bandASAL3,fuSAL3);
[Y X] = size(bandmatch);
weightmatrixA = zeros(Y,X);
weightmatrixB = zeros(Y,X);
 alpha = .9
  for i = 1:Y
    for j = 1:X
      if bandmatch(i,j) < alpha
        if bandASAL3(i,j) > fuSAL3(i,j)
          weightmatrixA(i,j) = 1;
        else
          weightmatrixB(i,j) = 1;
        end
      else
        wmin = .5 - .5*((1-bandmatch(i,j))/(1-alpha));
        wmax = 1 - wmin;

        if bandASAL3(i,j) > fuSAL3(i,j)
          weightmatrixA(i,j) = wmax;
          weightmatrixB(i,j) = wmin;
        else
          weightmatrixA(i,j) = wmin;
          weightmatrixB(i,j) = wmax;
        end

  end
end
end

 fuD33 = (weightmatrixA .* bandAD33) +  (weightmatrixB .* fuD33);

clear weightmatrixA bandAD33 weightmatrixB bandmatch fuSAL3
clear bandASAL3


bandASAL4 = salience(bandAD34);
fuSAL4 = salience(fuD34);
bandmatch = match(bandAD34,fuD34,bandASAL4,fuSAL4);
[Y X] = size(bandmatch);
weightmatrixA = zeros(Y,X);
weightmatrixB = zeros(Y,X);
```

```
alpha = .9
  for i = 1:Y
    for j = 1:X
      if bandmatch(i,j) < alpha
        if bandASAL4(i,j) > fuSAL4(i,j)
          weightmatrixA(i,j) = 1;
        else
          weightmatrixB(i,j) = 1;
        end
      else
        wmin = .5 - .5*((1-bandmatch(i,j))/(1-alpha));
        wmax = 1 - wmin;

        if bandASAL4(i,j) > fuSAL4(i,j)
          weightmatrixA(i,j) = wmax;
          weightmatrixB(i,j) = wmin;
        else
          weightmatrixA(i,j) = wmin;
          weightmatrixB(i,j) = wmax;
        end

  end
end
end

 fuD34 = (weightmatrixA .* bandAD34) +  (weightmatrixB .* fuD34);

clear weightmatrixA bandAD34 weightmatrixB bandmatch fuSAL4
clear bandASAL4
end

[OLD31 OLD32 OLD33 OLD34] = OLP(fuD31, fuD32, fuD33, fuD34,4);
clear fuD31 fuD32 fuD33 fuD34

[FSD3] = FSDLP(OLD31, OLD32, OLD33, OLD34);
clear OLD31 OLD32 OLD33 OLD34

[RELAP3] = RELP(FSD3,W);
clear FSD3



[fuD41 fuD42 fuD43 fuD44] = Orient_Grad_Pyramid(fuG4,w,4);
fuG5 = Grad_reduce(fuG4,W);
clear fuG4


for band = 332:333
bandAG0 = ennormal(getim512(band));

bandAG1 = Grad_reduce(bandAG0,W);
clear bandAG0
bandAG2 = Grad_reduce(bandAG1,W);
clear bandAG1
bandAG3 = Grad_reduce(bandAG2,W);
clear bandAG2
bandAG4 = Grad_reduce(bandAG3,W);
clear bandAG3

[bandAD41 bandAD42 bandAD43 bandAD44] = Orient_Grad_Pyramid(bandAG4,w,4);
clear bandAG4

bandASAL1 = salience(bandAD41);
fuSAL1 = salience(fuD41);
bandmatch = match(bandAD41,fuD41,bandASAL1,fuSAL1);
```

```
[Y X] = size(bandmatch);
weightmatrixA = zeros(Y,X);
weightmatrixB = zeros(Y,X);
 alpha = .9
   for i = 1:Y
     for j = 1:X
        if bandmatch(i,j) < alpha
           if bandASAL1(i,j) > fuSAL1(i,j)
              weightmatrixA(i,j) = 1;
           else
              weightmatrixB(i,j) = 1;
           end %if else
        else
           wmin = .5 - .5*((1-bandmatch(i,j))/(1-alpha));
           wmax = 1 - wmin;

           if bandASAL1(i,j) > fuSAL1(i,j)
              weightmatrixA(i,j) = wmax;
              weightmatrixB(i,j) = wmin;
           else
              weightmatrixA(i,j) = wmin;
              weightmatrixB(i,j) = wmax;
           end %end if else

        end %end if else
     end
  end
 fuD41 = (weightmatrixA .* bandAD41) +  (weightmatrixB .* fuD41);

clear weightmatrixA bandAD41 weightmatrixB bandmatch fuSAL1
clear bandASAL1

bandASAL2 = salience(bandAD42);
fuSAL2 = salience(fuD42);
bandmatch = match(bandAD42,fuD42,bandASAL2,fuSAL2);
[Y X] = size(bandmatch);
weightmatrixA = zeros(Y,X);
weightmatrixB = zeros(Y,X);
 alpha = .9
   for i = 1:Y
     for j = 1:X
        if bandmatch(i,j) < alpha
           if bandASAL2(i,j) > fuSAL2(i,j)
              weightmatrixA(i,j) = 1;
           else
              weightmatrixB(i,j) = 1;
           end
        else
           wmin = .5 - .5*((1-bandmatch(i,j))/(1-alpha));
           wmax = 1 - wmin;

           if bandASAL2(i,j) > fuSAL2(i,j)
              weightmatrixA(i,j) = wmax;
              weightmatrixB(i,j) = wmin;
           else
              weightmatrixA(i,j) = wmin;
              weightmatrixB(i,j) = wmax;
           end

   end
end
end

 fuD42 = (weightmatrixA .* bandAD42) +  (weightmatrixB .* fuD42);
```

```
clear weightmatrixA bandAD42 weightmatrixB bandmatch fuSAL2
clear bandASAL2

bandASAL3 = salience(bandAD43);
fuSAL3 = salience(fuD43);
bandmatch = match(bandAD43,fuD43,bandASAL3,fuSAL3);
[Y X] = size(bandmatch);
weightmatrixA = zeros(Y,X);
weightmatrixB = zeros(Y,X);
 alpha = .9
   for i = 1:Y
     for j = 1:X
        if bandmatch(i,j) < alpha
           if bandASAL3(i,j) > fuSAL3(i,j)
              weightmatrixA(i,j) = 1;
           else
              weightmatrixB(i,j) = 1;
           end
        else
           wmin = .5 - .5*((1-bandmatch(i,j))/(1-alpha));
           wmax = 1 - wmin;

           if bandASAL3(i,j) > fuSAL3(i,j)
              weightmatrixA(i,j) = wmax;
              weightmatrixB(i,j) = wmin;
           else
              weightmatrixA(i,j) = wmin;
              weightmatrixB(i,j) = wmax;
           end

   end
end
end

 fuD43 = (weightmatrixA .* bandAD43) +  (weightmatrixB .* fuD43);

clear weightmatrixA bandAD43 weightmatrixB bandmatch fuSAL3
clear bandASAL3


bandASAL4 = salience(bandAD44);
fuSAL4 = salience(fuD44);
bandmatch = match(bandAD44,fuD44,bandASAL4,fuSAL4);
[Y X] = size(bandmatch);
weightmatrixA = zeros(Y,X);
weightmatrixB = zeros(Y,X);
 alpha = .9
   for i = 1:Y
     for j = 1:X
        if bandmatch(i,j) < alpha
           if bandASAL4(i,j) > fuSAL4(i,j)
              weightmatrixA(i,j) = 1;
           else
              weightmatrixB(i,j) = 1;
           end
        else
           wmin = .5 - .5*((1-bandmatch(i,j))/(1-alpha));
           wmax = 1 - wmin;

           if bandASAL4(i,j) > fuSAL4(i,j)
              weightmatrixA(i,j) = wmax;
              weightmatrixB(i,j) = wmin;
           else
              weightmatrixA(i,j) = wmin;
              weightmatrixB(i,j) = wmax;
```

```
         end

   end
end
end

 fuD44 = (weightmatrixA .* bandAD44) +  (weightmatrixB .* fuD44);

clear weightmatrixA bandAD44 weightmatrixB bandmatch fuSAL4
clear bandASAL4

end %for loop

[OLD41 OLD42 OLD43 OLD44] = OLP(fuD41, fuD42, fuD43, fuD44,4);
clear fuD41 fuD42 fuD43 fuD44

[FSD4 ] = FSDLP(OLD41, OLD42, OLD43, OLD44);
clear OLD41 OLD42 OLD43 OLD44

[RELAP4 ] = RELP(FSD4,W);
clear FSD4



[fuD51 fuD52 fuD53 fuD54] = Orient_Grad_Pyramid(fuG5,w,4);
fuG6 = Grad_reduce(fuG5,W);
clear fuG5


for band = 332:333
bandAG0 = ennormal(getim512(band));

bandAG1 = Grad_reduce(bandAG0,W);
clear bandAG0
bandAG2 = Grad_reduce(bandAG1,W);
clear bandAG1
bandAG3 = Grad_reduce(bandAG2,W);
clear bandAG2
bandAG4 = Grad_reduce(bandAG3,W);
clear bandAG3
bandAG5 = Grad_reduce(bandAG4,W);
clear bandAG4
[bandAD51 bandAD52 bandAD53 bandAD54] = Orient_Grad_Pyramid(bandAG5,w,4);
clear bandAG5

bandASAL1 = salience(bandAD51);
fuSAL1 = salience(fuD51);
bandmatch = match(bandAD51,fuD51,bandASAL1,fuSAL1);
[Y X] = size(bandmatch);
weightmatrixA = zeros(Y,X);
weightmatrixB = zeros(Y,X);
 alpha = .9
   for i = 1:Y
     for j = 1:X
        if bandmatch(i,j) < alpha
           if bandASAL1(i,j) > fuSAL1(i,j)
              weightmatrixA(i,j) = 1;
           else
              weightmatrixB(i,j) = 1;
           end %if else
        else
           wmin = .5 - .5*((1-bandmatch(i,j))/(1-alpha));
           wmax = 1 - wmin;

           if bandASAL1(i,j) > fuSAL1(i,j)
```

```
            weightmatrixA(i,j) = wmax;
            weightmatrixB(i,j) = wmin;
          else
            weightmatrixA(i,j) = wmin;
            weightmatrixB(i,j) = wmax;
          end %end if else

        end %end if else
      end
    end
fuD51 = (weightmatrixA .* bandAD51) +  (weightmatrixB .* fuD51);

clear weightmatrixA bandAD51 weightmatrixB bandmatch fuSAL1
clear bandASAL1

bandASAL2 = salience(bandAD52);
fuSAL2 = salience(fuD52);
bandmatch = match(bandAD52,fuD52,bandASAL2,fuSAL2);
[Y X] = size(bandmatch);
weightmatrixA = zeros(Y,X);
weightmatrixB = zeros(Y,X);
alpha = .9
  for i = 1:Y
    for j = 1:X
      if bandmatch(i,j) < alpha
        if bandASAL2(i,j) > fuSAL2(i,j)
          weightmatrixA(i,j) = 1;
        else
          weightmatrixB(i,j) = 1;
        end
      else
        wmin = .5 - .5*((1-bandmatch(i,j))/(1-alpha));
        wmax = 1 - wmin;

        if bandASAL2(i,j) > fuSAL2(i,j)
          weightmatrixA(i,j) = wmax;
          weightmatrixB(i,j) = wmin;
        else
          weightmatrixA(i,j) = wmin;
          weightmatrixB(i,j) = wmax;
        end

  end
end
end

fuD52 = (weightmatrixA .* bandAD52) +  (weightmatrixB .* fuD52);

clear weightmatrixA bandAD52 weightmatrixB bandmatch fuSAL2
clear bandASAL2

bandASAL3 = salience(bandAD53);
fuSAL3 = salience(fuD53);
bandmatch = match(bandAD53,fuD53,bandASAL3,fuSAL3);
[Y X] = size(bandmatch);
weightmatrixA = zeros(Y,X);
weightmatrixB = zeros(Y,X);
alpha = .9
  for i = 1:Y
    for j = 1:X
      if bandmatch(i,j) < alpha
        if bandASAL3(i,j) > fuSAL3(i,j)
          weightmatrixA(i,j) = 1;
        else
          weightmatrixB(i,j) = 1;
```

```
          end
        else
          wmin = .5 - .5*((1-bandmatch(i,j))/(1-alpha));
          wmax = 1 - wmin;

          if bandASAL3(i,j) > fuSAL3(i,j)
            weightmatrixA(i,j) = wmax;
            weightmatrixB(i,j) = wmin;
          else
            weightmatrixA(i,j) = wmin;
            weightmatrixB(i,j) = wmax;
          end

   end
end
end

 fuD53 = (weightmatrixA .* bandAD53) +  (weightmatrixB .* fuD53);

clear weightmatrixA bandAD53 weightmatrixB bandmatch fuSAL3
clear bandASAL3


bandASAL4 = salience(bandAD54);
fuSAL4 = salience(fuD54);
bandmatch = match(bandAD54,fuD54,bandASAL4,fuSAL4);
[Y X] = size(bandmatch);
weightmatrixA = zeros(Y,X);
weightmatrixB = zeros(Y,X);
 alpha = .9
   for i = 1:Y
     for j = 1:X
        if bandmatch(i,j) < alpha
          if bandASAL4(i,j) > fuSAL4(i,j)
            weightmatrixA(i,j) = 1;
          else
            weightmatrixB(i,j) = 1;
          end
        else
          wmin = .5 - .5*((1-bandmatch(i,j))/(1-alpha));
          wmax = 1 - wmin;

          if bandASAL4(i,j) > fuSAL4(i,j)
            weightmatrixA(i,j) = wmax;
            weightmatrixB(i,j) = wmin;
          else
            weightmatrixA(i,j) = wmin;
            weightmatrixB(i,j) = wmax;
          end

   end
end
end

 fuD54 = (weightmatrixA .* bandAD54) +  (weightmatrixB .* fuD54);

clear weightmatrixA bandAD54 weightmatrixB bandmatch fuSAL4
clear bandASAL4

end %for loop

[OLD51 OLD52 OLD53 OLD54] = OLP(fuD51, fuD52, fuD53, fuD54,4);
clear fuD51 fuD52 fuD53 fuD54

[FSD5 ] = FSDLP(OLD51, OLD52, OLD53, OLD54);
```

```
clear OLD51 OLD52 OLD53 OLD54

[RELAP5 ] = RELP(FSD5,W);
clear FSD5


% Now the reconstruction of the Image begins.  Using the results of
% equation (A3) Note: I use the original top level from the Gaussian
% Pyramid to start the reconstruction.  i.e. G6 = bandAG6
band231G0 = ennormal(getim512(331));
band231G1 = Grad_reduce(band231G0,W);
clear band231G0
band231G2 = Grad_reduce(band231G1,W);
clear band231G1
band231G3 = Grad_reduce(band231G2,W);
clear band231G2
band231G4 = Grad_reduce(band231G3,W);
clear band231G3
band231G5 = Grad_reduce(band231G4,W);
clear band231G4
G6 = Grad_reduce(band231G5,W);
clear band231G5

G5 = RELAP5 + Gauss_expand(G6,W);
clear RELAP5 G6

G4 = RELAP4 + Gauss_expand(G5,W);
clear RELAP4 G5

G3 = RELAP3 + Gauss_expand(G4,W);
clear RELAP3 G4

G2 = RELAP2 + Gauss_expand(G3,W);
clear RELAP2 G3

G1 = RELAP1 + Gauss_expand(G2,W);
clear RELAP1 G2

G0 = clean512(RELAP + Gauss_expand(G1,W));
clear RELAP G1

save  blenna62 G0

quit
```

E.3 Image Fusion Algorithm For SAR Imagery

```
%This program fuses three SAR images.  The resulting output is a single image that has the same
% dimensions as the input images, but has now fused all  three.   Each input  image is padded and
% energy normalized before it is deconstructed.  It performs  6 levels of deconstruction and the first
% SAR image is  chosen for the gross approximation.  The threshold value is .90.

weights

fuG0 = getim_sar('km15hhl');

[fuD01 fuD02 fuD03 fuD04] = Orient_Grad_Pyramid(fuG0,w,4);
fuG1 = Grad_reduce(fuG0,W);
clear fuG0

for band = 60:30:90

if (band == 60)
```

```
      bandAG0 = getim_sar('km15hv1');
else

bandAG0 = getim_sar('km15vv1');
end

[bandAD01 bandAD02 bandAD03 bandAD04] = Orient_Grad_Pyramid(bandAG0,w,4);

bandASAL1 = salience(bandAD01);
fuSAL1 = salience(fuD01);
bandmatch = match(bandAD01,fuD01,bandASAL1,fuSAL1);
[Y X] = size(bandmatch);
weightmatrixA = zeros(Y,X);
weightmatrixB = zeros(Y,X);
 alpha = .9
   for i = 1:Y
     for j = 1:X
        if bandmatch(i,j) < alpha
           if bandASAL1(i,j) > fuSAL1(i,j)
              weightmatrixA(i,j) = 1;
           else
              weightmatrixB(i,j) = 1;
           end %if else
        else
           wmin = .5 - .5*((1-bandmatch(i,j))/(1-alpha));
           wmax = 1 - wmin;

           if bandASAL1(i,j) > fuSAL1(i,j)
              weightmatrixA(i,j) = wmax;
              weightmatrixB(i,j) = wmin;
           else
              weightmatrixA(i,j) = wmin;
              weightmatrixB(i,j) = wmax;
           end %end if else

        end %end if else
     end
  end
 fuD01 = (weightmatrixA .* bandAD01) +  (weightmatrixB .* fuD01);

clear weightmatrixA bandAD01 weightmatrixB bandmatch fuSAL1
clear bandASAL1

bandASAL2 = salience(bandAD02);
fuSAL2 = salience(fuD02);
bandmatch = match(bandAD02,fuD02,bandASAL2,fuSAL2);
[Y X] = size(bandmatch);
weightmatrixA = zeros(Y,X);
weightmatrixB = zeros(Y,X);
 alpha = .9
   for i = 1:Y
     for j = 1:X
        if bandmatch(i,j) < alpha
           if bandASAL2(i,j) > fuSAL2(i,j)
              weightmatrixA(i,j) = 1;
           else
              weightmatrixB(i,j) = 1;
           end
        else
           wmin = .5 - .5*((1-bandmatch(i,j))/(1-alpha));
           wmax = 1 - wmin;

           if bandASAL2(i,j) > fuSAL2(i,j)
              weightmatrixA(i,j) = wmax;
```

```
            weightmatrixB(i,j) = wmin;
          else
            weightmatrixA(i,j) = wmin;
            weightmatrixB(i,j) = wmax;
          end

   end
end
end

 fuD02 = (weightmatrixA .* bandAD02) +  (weightmatrixB .* fuD02);

clear weightmatrixA bandAD02 weightmatrixB bandmatch fuSAL2
clear bandASAL2

bandASAL3 = salience(bandAD03);
fuSAL3 = salience(fuD03);
bandmatch = match(bandAD03,fuD03,bandASAL3,fuSAL3);
[Y X] = size(bandmatch);
weightmatrixA = zeros(Y,X);
weightmatrixB = zeros(Y,X);
 alpha = .9
   for i = 1:Y
     for j = 1:X
        if bandmatch(i,j) < alpha
          if bandASAL3(i,j) > fuSAL3(i,j)
            weightmatrixA(i,j) = 1;
          else
            weightmatrixB(i,j) = 1;
          end
        else
          wmin = .5 - .5*((1-bandmatch(i,j))/(1-alpha));
          wmax = 1 - wmin;

          if bandASAL3(i,j) > fuSAL3(i,j)
            weightmatrixA(i,j) = wmax;
            weightmatrixB(i,j) = wmin;
          else
            weightmatrixA(i,j) = wmin;
            weightmatrixB(i,j) = wmax;
          end

   end
end
end

 fuD03 = (weightmatrixA .* bandAD03) +  (weightmatrixB .* fuD03);

clear weightmatrixA bandAD03 weightmatrixB bandmatch fuSAL3
clear bandASAL3


bandASAL4 = salience(bandAD04);
fuSAL4 = salience(fuD04);
bandmatch = match(bandAD04,fuD04,bandASAL4,fuSAL4);
[Y X] = size(bandmatch);
weightmatrixA = zeros(Y,X);
weightmatrixB = zeros(Y,X);
 alpha = .9
   for i = 1:Y
     for j = 1:X
        if bandmatch(i,j) < alpha
          if bandASAL4(i,j) > fuSAL4(i,j)
            weightmatrixA(i,j) = 1;
          else
```

```
            weightmatrixB(i,j) = 1;
          end
        else
          wmin = .5 - .5*((1-bandmatch(i,j))/(1-alpha));
          wmax = 1 - wmin;

          if bandASAL4(i,j) > fuSAL4(i,j)
            weightmatrixA(i,j) = wmax;
            weightmatrixB(i,j) = wmin;
          else
            weightmatrixA(i,j) = wmin;
            weightmatrixB(i,j) = wmax;
          end

    end
end
end

 fuD04 = (weightmatrixA .* bandAD04) +  (weightmatrixB .* fuD04);

clear weightmatrixA bandAD04 weightmatrixB bandmatch fuSAL4
clear bandASAL4

end %for loop

[OLD01 OLD02 OLD03 OLD04] = OLP(fuD01, fuD02, fuD03, fuD04,4);
clear fuD01 fuD02 fuD03 fuD04

[FSD ] = FSDLP(OLD01, OLD02, OLD03, OLD04);
clear OLD01 OLD02 OLD03 OLD04

[RELAP ] = RELP(FSD,W);
clear FSD


[fuD11 fuD12 fuD13 fuD14] = Orient_Grad_Pyramid(fuG1,w,4);
fuG2 = Grad_reduce(fuG1,W);
clear fuG1

for band = 60:30:90,

if (band == 60)

       bandAG0 = getim_sar('km15hv1');
else

bandAG0 = getim_sar('km15vv1');
end

bandAG1 = Grad_reduce(bandAG0,W);
clear bandAG0

[bandAD11 bandAD12 bandAD13 bandAD14] = Orient_Grad_Pyramid(bandAG1,w,4);

bandASAL1 = salience(bandAD11);
fuSAL1 = salience(fuD11);
bandmatch = match(bandAD11,fuD11,bandASAL1,fuSAL1);
[Y X] = size(bandmatch);
weightmatrixA = zeros(Y,X);
weightmatrixB = zeros(Y,X);
 alpha = .9
   for i = 1:Y
     for j = 1:X
        if bandmatch(i,j) < alpha
          if bandASAL1(i,j) > fuSAL1(i,j)
```

```
            weightmatrixA(i,j) = 1;
          else
            weightmatrixB(i,j) = 1;
          end
        else
          wmin = .5 - .5*((1-bandmatch(i,j))/(1-alpha));
          wmax = 1 - wmin;

          if bandASAL1(i,j) > fuSAL1(i,j)
            weightmatrixA(i,j) = wmax;
            weightmatrixB(i,j) = wmin;
          else
            weightmatrixA(i,j) = wmin;
            weightmatrixB(i,j) = wmax;
          end

    end
end
end

 fuD11 = (weightmatrixA .* bandAD11) +  (weightmatrixB .* fuD11);

clear weightmatrixA bandAD11 weightmatrixB bandmatch fuSAL1
clear bandASAL1

bandASAL2 = salience(bandAD12);
fuSAL2 = salience(fuD12);
bandmatch = match(bandAD12,fuD12,bandASAL2,fuSAL2);
[Y X] = size(bandmatch);
weightmatrixA = zeros(Y,X);
weightmatrixB = zeros(Y,X);
 alpha = .9
   for i = 1:Y
     for j = 1:X
        if bandmatch(i,j) < alpha
          if bandASAL2(i,j) > fuSAL2(i,j)
            weightmatrixA(i,j) = 1;
          else
            weightmatrixB(i,j) = 1;
          end
        else
          wmin = .5 - .5*((1-bandmatch(i,j))/(1-alpha));
          wmax = 1 - wmin;

          if bandASAL2(i,j) > fuSAL2(i,j)
            weightmatrixA(i,j) = wmax;
            weightmatrixB(i,j) = wmin;
          else
            weightmatrixA(i,j) = wmin;
            weightmatrixB(i,j) = wmax;
          end

    end
end
end

 fuD12 = (weightmatrixA .* bandAD12) +  (weightmatrixB .* fuD12);

clear weightmatrixA bandAD12 weightmatrixB bandmatch fuSAL2
clear bandASAL2

bandASAL3 = salience(bandAD13);
fuSAL3 = salience(fuD13);
bandmatch = match(bandAD13,fuD13,bandASAL3,fuSAL3);
[Y X] = size(bandmatch);
```

```
weightmatrixA = zeros(Y,X);
weightmatrixB = zeros(Y,X);
 alpha = .9
   for i = 1:Y
     for j = 1:X
        if bandmatch(i,j) < alpha
           if bandASAL3(i,j) > fuSAL3(i,j)
              weightmatrixA(i,j) = 1;
           else
              weightmatrixB(i,j) = 1;
           end
        else
           wmin = .5 - .5*((1-bandmatch(i,j))/(1-alpha));
           wmax = 1 - wmin;

           if bandASAL3(i,j) > fuSAL3(i,j)
              weightmatrixA(i,j) = wmax;
              weightmatrixB(i,j) = wmin;
           else
              weightmatrixA(i,j) = wmin;
              weightmatrixB(i,j) = wmax;
           end

   end
end
end

 fuD13 = (weightmatrixA .* bandAD13) +  (weightmatrixB .* fuD13);

clear weightmatrixA bandAD13 weightmatrixB bandmatch fuSAL3
clear bandASAL3


bandASAL4 = salience(bandAD14);
fuSAL4 = salience(fuD14);
bandmatch = match(bandAD14,fuD14,bandASAL4,fuSAL4);
[Y X] = size(bandmatch);
weightmatrixA = zeros(Y,X);
weightmatrixB = zeros(Y,X);
 alpha = .9
   for i = 1:Y
     for j = 1:X
        if bandmatch(i,j) < alpha
           if bandASAL4(i,j) > fuSAL4(i,j)
              weightmatrixA(i,j) = 1;
           else
              weightmatrixB(i,j) = 1;
           end
        else
           wmin = .5 - .5*((1-bandmatch(i,j))/(1-alpha));
           wmax = 1 - wmin;

           if bandASAL4(i,j) > fuSAL4(i,j)
              weightmatrixA(i,j) = wmax;
              weightmatrixB(i,j) = wmin;
           else
              weightmatrixA(i,j) = wmin;
              weightmatrixB(i,j) = wmax;
           end

   end
end
end

 fuD14 = (weightmatrixA .* bandAD14) +  (weightmatrixB .* fuD14);
```

```
clear weightmatrixA bandAD14 weightmatrixB bandmatch fuSAL4
clear bandASAL4

end

[OLD11 OLD12 OLD13 OLD14] = OLP(fuD11, fuD12, fuD13, fuD14,4);
clear fuD11 fuD12 fuD13 fuD14

[FSD1] = FSDLP(OLD11, OLD12, OLD13, OLD14);
clear OLD11 OLD12 OLD13 OLD14

[RELAP1] = RELP(FSD1,W);
clear FSD1


[fuD21 fuD22 fuD23 fuD24] = Orient_Grad_Pyramid(fuG2,w,4);
fuG3 = Grad_reduce(fuG2,W);
clear fuG2


for band = 60:30:90,

if (band == 60)

       bandAG0 = getim_sar('km15hv1');
else

bandAG0 = getim_sar('km15vv1');
end

bandAG1 = Grad_reduce(bandAG0,W);
clear bandAG0
bandAG2 = Grad_reduce(bandAG1,W);
clear bandAG1


[bandAD21 bandAD22 bandAD23 bandAD24] = Orient_Grad_Pyramid(bandAG2,w,4);
bandAG3 = Grad_reduce(bandAG2,W);
clear bandAG2

bandASAL1 = salience(bandAD21);
fuSAL1 = salience(fuD21);
bandmatch = match(bandAD21,fuD21,bandASAL1,fuSAL1);
[Y X] = size(bandmatch);
weightmatrixA = zeros(Y,X);
weightmatrixB = zeros(Y,X);
 alpha = .9
   for i = 1:Y
     for j = 1:X
        if bandmatch(i,j) < alpha
           if bandASAL1(i,j) > fuSAL1(i,j)
              weightmatrixA(i,j) = 1;
           else
              weightmatrixB(i,j) = 1;
           end
        else
           wmin = .5 - .5*((1-bandmatch(i,j))/(1-alpha));
           wmax = 1 - wmin;

           if bandASAL1(i,j) > fuSAL1(i,j)
              weightmatrixA(i,j) = wmax;
              weightmatrixB(i,j) = wmin;
           else
              weightmatrixA(i,j) = wmin;
```

```
            weightmatrixB(i,j) = wmax;
          end

   end
end
end

 fuD21 = (weightmatrixA .* bandAD21) +  (weightmatrixB .* fuD21);

clear weightmatrixA bandAD21 weightmatrixB bandmatch fuSAL1
clear bandASAL1

bandASAL2 = salience(bandAD22);
fuSAL2 = salience(fuD22);
bandmatch = match(bandAD22,fuD22,bandASAL2,fuSAL2);
[Y X] = size(bandmatch);
weightmatrixA = zeros(Y,X);
weightmatrixB = zeros(Y,X);
 alpha = .9
   for i = 1:Y
     for j = 1:X
        if bandmatch(i,j) < alpha
           if bandASAL2(i,j) > fuSAL2(i,j)
              weightmatrixA(i,j) = 1;
           else
              weightmatrixB(i,j) = 1;
           end
        else
           wmin = .5 - .5*((1-bandmatch(i,j))/(1-alpha));
           wmax = 1 - wmin;

           if bandASAL2(i,j) > fuSAL2(i,j)
              weightmatrixA(i,j) = wmax;
              weightmatrixB(i,j) = wmin;
           else
              weightmatrixA(i,j) = wmin;
              weightmatrixB(i,j) = wmax;
           end

   end
end
end

 fuD22 = (weightmatrixA .* bandAD22) +  (weightmatrixB .* fuD22);

clear weightmatrixA bandAD22 weightmatrixB bandmatch fuSAL2
clear bandASAL2

bandASAL3 = salience(bandAD23);
fuSAL3 = salience(fuD23);
bandmatch = match(bandAD23,fuD23,bandASAL3,fuSAL3);
[Y X] = size(bandmatch);
weightmatrixA = zeros(Y,X);
weightmatrixB = zeros(Y,X);
 alpha = .9
   for i = 1:Y
     for j = 1:X
        if bandmatch(i,j) < alpha
           if bandASAL3(i,j) > fuSAL3(i,j)
              weightmatrixA(i,j) = 1;
           else
              weightmatrixB(i,j) = 1;
           end
        else
           wmin = .5 - .5*((1-bandmatch(i,j))/(1-alpha));
```

```
          wmax = 1 - wmin;

          if bandASAL3(i,j) > fuSAL3(i,j)
            weightmatrixA(i,j) = wmax;
            weightmatrixB(i,j) = wmin;
          else
            weightmatrixA(i,j) = wmin;
            weightmatrixB(i,j) = wmax;
          end

   end
end
end

 fuD23 = (weightmatrixA .* bandAD23) +  (weightmatrixB .* fuD23);

clear weightmatrixA bandAD23 weightmatrixB bandmatch fuSAL3
clear bandASAL3


bandASAL4 = salience(bandAD24);
fuSAL4 = salience(fuD24);
bandmatch = match(bandAD24,fuD24,bandASAL4,fuSAL4);
[Y X] = size(bandmatch);
weightmatrixA = zeros(Y,X);
weightmatrixB = zeros(Y,X);
 alpha = .9
   for i = 1:Y
     for j = 1:X
        if bandmatch(i,j) < alpha
          if bandASAL4(i,j) > fuSAL4(i,j)
            weightmatrixA(i,j) = 1;
          else
            weightmatrixB(i,j) = 1;
          end
        else
          wmin = .5 - .5*((1-bandmatch(i,j))/(1-alpha));
          wmax = 1 - wmin;

          if bandASAL4(i,j) > fuSAL4(i,j)
            weightmatrixA(i,j) = wmax;
            weightmatrixB(i,j) = wmin;
          else
            weightmatrixA(i,j) = wmin;
            weightmatrixB(i,j) = wmax;
          end

   end
end
end

 fuD24 = (weightmatrixA .* bandAD24) +  (weightmatrixB .* fuD24);

clear weightmatrixA bandAD24 weightmatrixB bandmatch fuSAL4
clear bandASAL4

end

[OLD21 OLD22 OLD23 OLD24] = OLP(fuD21, fuD22, fuD23, fuD24,4);
clear fuD21 fuD22 fuD23 fuD24

[FSD2] = FSDLP(OLD21, OLD22, OLD23, OLD24);
clear OLD21 OLD22 OLD23 OLD24

[RELAP2] = RELP(FSD2,W);
```

```
clear FSD2

[fuD31 fuD32 fuD33 fuD34] = Orient_Grad_Pyramid(fuG3,w,4);
fuG4 = Grad_reduce(fuG3,W);
clear fuG3


for band = 60:30:90,


if (band == 60)

       bandAG0 = getim_sar('km15hvl');
else

bandAG0 = getim_sar('km15vvl');
end

bandAG1 = Grad_reduce(bandAG0,W);
clear bandAG0
bandAG2 = Grad_reduce(bandAG1,W);
clear bandAG1
bandAG3 = Grad_reduce(bandAG2,W);
clear bandAG2


[bandAD31 bandAD32 bandAD33 bandAD34] = Orient_Grad_Pyramid(bandAG3,w,4);
clear bandAG3

bandASAL1 = salience(bandAD31);
fuSAL1 = salience(fuD31);
bandmatch = match(bandAD31,fuD31,bandASAL1,fuSAL1);
[Y X] = size(bandmatch);
weightmatrixA = zeros(Y,X);
weightmatrixB = zeros(Y,X);
 alpha = .9
   for i = 1:Y
     for j = 1:X
        if bandmatch(i,j) < alpha
           if bandASAL1(i,j) > fuSAL1(i,j)
              weightmatrixA(i,j) = 1;
           else
              weightmatrixB(i,j) = 1;
           end
        else
           wmin = .5 - .5*((1-bandmatch(i,j))/(1-alpha));
           wmax = 1 - wmin;

           if bandASAL1(i,j) > fuSAL1(i,j)
              weightmatrixA(i,j) = wmax;
              weightmatrixB(i,j) = wmin;
           else
              weightmatrixA(i,j) = wmin;
              weightmatrixB(i,j) = wmax;
           end

   end
end
end

 fuD31 = (weightmatrixA .* bandAD31) +  (weightmatrixB .* fuD31);

clear weightmatrixA bandAD31 weightmatrixB bandmatch fuSAL1
clear bandASAL1
```

```
bandASAL2 = salience(bandAD32);
fuSAL2 = salience(fuD32);
bandmatch = match(bandAD32,fuD32,bandASAL2,fuSAL2);
[Y X] = size(bandmatch);
weightmatrixA = zeros(Y,X);
weightmatrixB = zeros(Y,X);
 alpha = .9
   for i = 1:Y
     for j = 1:X
        if bandmatch(i,j) < alpha
           if bandASAL2(i,j) > fuSAL2(i,j)
             weightmatrixA(i,j) = 1;
           else
             weightmatrixB(i,j) = 1;
           end
        else
           wmin = .5 - .5*((1-bandmatch(i,j))/(1-alpha));
           wmax = 1 - wmin;

           if bandASAL2(i,j) > fuSAL2(i,j)
             weightmatrixA(i,j) = wmax;
             weightmatrixB(i,j) = wmin;
           else
             weightmatrixA(i,j) = wmin;
             weightmatrixB(i,j) = wmax;
           end

   end
end
end

 fuD32 = (weightmatrixA .* bandAD32) +  (weightmatrixB .* fuD32);

clear weightmatrixA bandAD32 weightmatrixB bandmatch fuSAL2
clear bandASAL2

bandASAL3 = salience(bandAD33);
fuSAL3 = salience(fuD33);
bandmatch = match(bandAD33,fuD33,bandASAL3,fuSAL3);
[Y X] = size(bandmatch);
weightmatrixA = zeros(Y,X);
weightmatrixB = zeros(Y,X);
 alpha = .9
   for i = 1:Y
     for j = 1:X
        if bandmatch(i,j) < alpha
           if bandASAL3(i,j) > fuSAL3(i,j)
             weightmatrixA(i,j) = 1;
           else
             weightmatrixB(i,j) = 1;
           end
        else
           wmin = .5 - .5*((1-bandmatch(i,j))/(1-alpha));
           wmax = 1 - wmin;

           if bandASAL3(i,j) > fuSAL3(i,j)
             weightmatrixA(i,j) = wmax;
             weightmatrixB(i,j) = wmin;
           else
             weightmatrixA(i,j) = wmin;
             weightmatrixB(i,j) = wmax;
           end

   end
end
```

```
end

 fuD33 = (weightmatrixA .* bandAD33) +  (weightmatrixB .* fuD33);

clear weightmatrixA bandAD33 weightmatrixB bandmatch fuSAL3
clear bandASAL3


bandASAL4 = salience(bandAD34);
fuSAL4 = salience(fuD34);
bandmatch = match(bandAD34,fuD34,bandASAL4,fuSAL4);
[Y X] = size(bandmatch);
weightmatrixA = zeros(Y,X);
weightmatrixB = zeros(Y,X);
 alpha = .9
   for i = 1:Y
     for j = 1:X
        if bandmatch(i,j) < alpha
           if bandASAL4(i,j) > fuSAL4(i,j)
              weightmatrixA(i,j) = 1;
           else
              weightmatrixB(i,j) = 1;
           end
        else
           wmin = .5 - .5*((1-bandmatch(i,j))/(1-alpha));
           wmax = 1 - wmin;

           if bandASAL4(i,j) > fuSAL4(i,j)
              weightmatrixA(i,j) = wmax;
              weightmatrixB(i,j) = wmin;
           else
              weightmatrixA(i,j) = wmin;
              weightmatrixB(i,j) = wmax;
           end

   end
end
end

 fuD34 = (weightmatrixA .* bandAD34) +  (weightmatrixB .* fuD34);

clear weightmatrixA bandAD34 weightmatrixB bandmatch fuSAL4
clear bandASAL4
end

[OLD31 OLD32 OLD33 OLD34] = OLP(fuD31, fuD32, fuD33, fuD34,4);
clear fuD31 fuD32 fuD33 fuD34

[FSD3] = FSDLP(OLD31, OLD32, OLD33, OLD34);
clear OLD31 OLD32 OLD33 OLD34

[RELAP3] = RELP(FSD3,W);
clear FSD3



[fuD41 fuD42 fuD43 fuD44] = Orient_Grad_Pyramid(fuG4,w,4);
fuG5 = Grad_reduce(fuG4,W);
clear fuG4


for band = 60:30:90
if (band == 60)

       bandAG0 = getim_sar('km15hv1');
```

```
else

bandAG0 = getim_sar('km15vv1');
end

bandAG1 = Grad_reduce(bandAG0,W);
clear bandAG0
bandAG2 = Grad_reduce(bandAG1,W);
clear bandAG1
bandAG3 = Grad_reduce(bandAG2,W);
clear bandAG2
bandAG4 = Grad_reduce(bandAG3,W);
clear bandAG3

[bandAD41 bandAD42 bandAD43 bandAD44] = Orient_Grad_Pyramid(bandAG4,w,4);
clear bandAG4

bandASAL1 = salience(bandAD41);
fuSAL1 = salience(fuD41);
bandmatch = match(bandAD41,fuD41,bandASAL1,fuSAL1);
[Y X] = size(bandmatch);
weightmatrixA = zeros(Y,X);
weightmatrixB = zeros(Y,X);
 alpha = .9
   for i = 1:Y
     for j = 1:X
        if bandmatch(i,j) < alpha
           if bandASAL1(i,j) > fuSAL1(i,j)
              weightmatrixA(i,j) = 1;
           else
              weightmatrixB(i,j) = 1;
           end %if else
        else
           wmin = .5 - .5*((1-bandmatch(i,j))/(1-alpha));
           wmax = 1 - wmin;

           if bandASAL1(i,j) > fuSAL1(i,j)
              weightmatrixA(i,j) = wmax;
              weightmatrixB(i,j) = wmin;
           else
              weightmatrixA(i,j) = wmin;
              weightmatrixB(i,j) = wmax;
           end %end if else

        end %end if else
     end
  end
 fuD41 = (weightmatrixA .* bandAD41) +  (weightmatrixB .* fuD41);

clear weightmatrixA bandAD41 weightmatrixB bandmatch fuSAL1
clear bandASAL1

bandASAL2 = salience(bandAD42);
fuSAL2 = salience(fuD42);
bandmatch = match(bandAD42,fuD42,bandASAL2,fuSAL2);
[Y X] = size(bandmatch);
weightmatrixA = zeros(Y,X);
weightmatrixB = zeros(Y,X);
 alpha = .9
   for i = 1:Y
     for j = 1:X
        if bandmatch(i,j) < alpha
           if bandASAL2(i,j) > fuSAL2(i,j)
              weightmatrixA(i,j) = 1;
           else
```

```
            weightmatrixB(i,j) = 1;
          end
        else
          wmin = .5 - .5*((1-bandmatch(i,j))/(1-alpha));
          wmax = 1 - wmin;

          if bandASAL2(i,j) > fuSAL2(i,j)
            weightmatrixA(i,j) = wmax;
            weightmatrixB(i,j) = wmin;
          else
            weightmatrixA(i,j) = wmin;
            weightmatrixB(i,j) = wmax;
          end

    end
end
end

 fuD42 = (weightmatrixA .* bandAD42) +  (weightmatrixB .* fuD42);

clear weightmatrixA bandAD42 weightmatrixB bandmatch fuSAL2
clear bandASAL2

bandASAL3 = salience(bandAD43);
fuSAL3 = salience(fuD43);
bandmatch = match(bandAD43,fuD43,bandASAL3,fuSAL3);
[Y X] = size(bandmatch);
weightmatrixA = zeros(Y,X);
weightmatrixB = zeros(Y,X);
 alpha = .9
   for i = 1:Y
     for j = 1:X
        if bandmatch(i,j) < alpha
          if bandASAL3(i,j) > fuSAL3(i,j)
            weightmatrixA(i,j) = 1;
          else
            weightmatrixB(i,j) = 1;
          end
        else
          wmin = .5 - .5*((1-bandmatch(i,j))/(1-alpha));
          wmax = 1 - wmin;

          if bandASAL3(i,j) > fuSAL3(i,j)
            weightmatrixA(i,j) = wmax;
            weightmatrixB(i,j) = wmin;
          else
            weightmatrixA(i,j) = wmin;
            weightmatrixB(i,j) = wmax;
          end

    end
end
end

 fuD43 = (weightmatrixA .* bandAD43) +  (weightmatrixB .* fuD43);

clear weightmatrixA bandAD43 weightmatrixB bandmatch fuSAL3
clear bandASAL3


bandASAL4 = salience(bandAD44);
fuSAL4 = salience(fuD44);
bandmatch = match(bandAD44,fuD44,bandASAL4,fuSAL4);
[Y X] = size(bandmatch);
weightmatrixA = zeros(Y,X);
```

```
weightmatrixB = zeros(Y,X);
 alpha = .9
   for i = 1:Y
     for j = 1:X
        if bandmatch(i,j) < alpha
           if bandASAL4(i,j) > fuSAL4(i,j)
              weightmatrixA(i,j) = 1;
           else
              weightmatrixB(i,j) = 1;
           end
        else
           wmin = .5 - .5*((1-bandmatch(i,j))/(1-alpha));
           wmax = 1 - wmin;

           if bandASAL4(i,j) > fuSAL4(i,j)
              weightmatrixA(i,j) = wmax;
              weightmatrixB(i,j) = wmin;
           else
              weightmatrixA(i,j) = wmin;
              weightmatrixB(i,j) = wmax;
           end

   end
end
end

 fuD44 = (weightmatrixA .* bandAD44) +  (weightmatrixB .* fuD44);

clear weightmatrixA bandAD44 weightmatrixB bandmatch fuSAL4
clear bandASAL4

end %for loop

[OLD41 OLD42 OLD43 OLD44] = OLP(fuD41, fuD42, fuD43, fuD44,4);
clear fuD41 fuD42 fuD43 fuD44

[FSD4 ] = FSDLP(OLD41, OLD42, OLD43, OLD44);
clear OLD41 OLD42 OLD43 OLD44

[RELAP4 ] = RELP(FSD4,W);
clear FSD4



[fuD51 fuD52 fuD53 fuD54] = Orient_Grad_Pyramid(fuG5,w,4);
fuG6 = Grad_reduce(fuG5,W);
clear fuG5


for band = 60:30:90
if (band == 60)

       bandAG0 = getim_sar('km15hvl');
else

bandAG0 = getim_sar('km15vvl');
end

bandAG1 = Grad_reduce(bandAG0,W);
clear bandAG0
bandAG2 = Grad_reduce(bandAG1,W);
clear bandAG1
bandAG3 = Grad_reduce(bandAG2,W);
clear bandAG2
bandAG4 = Grad_reduce(bandAG3,W);
```

```
clear bandAG3
bandAG5 = Grad_reduce(bandAG4,W);
clear bandAG4
[bandAD51 bandAD52 bandAD53 bandAD54] = Orient_Grad_Pyramid(bandAG5,w,4);
clear bandAG5

bandASAL1 = salience(bandAD51);
fuSAL1 = salience(fuD51);
bandmatch = match(bandAD51,fuD51,bandASAL1,fuSAL1);
[Y X] = size(bandmatch);
weightmatrixA = zeros(Y,X);
weightmatrixB = zeros(Y,X);
 alpha = .9
   for i = 1:Y
     for j = 1:X
        if bandmatch(i,j) < alpha
           if bandASAL1(i,j) > fuSAL1(i,j)
              weightmatrixA(i,j) = 1;
           else
              weightmatrixB(i,j) = 1;
           end %if else
        else
           wmin = .5 - .5*((1-bandmatch(i,j))/(1-alpha));
           wmax = 1 - wmin;

           if bandASAL1(i,j) > fuSAL1(i,j)
              weightmatrixA(i,j) = wmax;
              weightmatrixB(i,j) = wmin;
           else
              weightmatrixA(i,j) = wmin;
              weightmatrixB(i,j) = wmax;
           end %end if else

        end %end if else
     end
  end
 fuD51 = (weightmatrixA .* bandAD51) +  (weightmatrixB .* fuD51);

clear weightmatrixA bandAD51 weightmatrixB bandmatch fuSAL1
clear bandASAL1

bandASAL2 = salience(bandAD52);
fuSAL2 = salience(fuD52);
bandmatch = match(bandAD52,fuD52,bandASAL2,fuSAL2);
[Y X] = size(bandmatch);
weightmatrixA = zeros(Y,X);
weightmatrixB = zeros(Y,X);
 alpha = .9
   for i = 1:Y
     for j = 1:X
        if bandmatch(i,j) < alpha
           if bandASAL2(i,j) > fuSAL2(i,j)
              weightmatrixA(i,j) = 1;
           else
              weightmatrixB(i,j) = 1;
           end
        else
           wmin = .5 - .5*((1-bandmatch(i,j))/(1-alpha));
           wmax = 1 - wmin;

           if bandASAL2(i,j) > fuSAL2(i,j)
              weightmatrixA(i,j) = wmax;
              weightmatrixB(i,j) = wmin;
           else
              weightmatrixA(i,j) = wmin;
```

```
            weightmatrixB(i,j) = wmax;
          end

   end
end
end

 fuD52 = (weightmatrixA .* bandAD52) +  (weightmatrixB .* fuD52);

clear weightmatrixA bandAD52 weightmatrixB bandmatch fuSAL2
clear bandASAL2

bandASAL3 = salience(bandAD53);
fuSAL3 = salience(fuD53);
bandmatch = match(bandAD53,fuD53,bandASAL3,fuSAL3);
[Y X] = size(bandmatch);
weightmatrixA = zeros(Y,X);
weightmatrixB = zeros(Y,X);
 alpha = .9
   for i = 1:Y
     for j = 1:X
        if bandmatch(i,j) < alpha
           if bandASAL3(i,j) > fuSAL3(i,j)
             weightmatrixA(i,j) = 1;
           else
             weightmatrixB(i,j) = 1;
           end
        else
           wmin = .5 - .5*((1-bandmatch(i,j))/(1-alpha));
           wmax = 1 - wmin;

           if bandASAL3(i,j) > fuSAL3(i,j)
             weightmatrixA(i,j) = wmax;
             weightmatrixB(i,j) = wmin;
           else
             weightmatrixA(i,j) = wmin;
             weightmatrixB(i,j) = wmax;
           end

   end
end
end

 fuD53 = (weightmatrixA .* bandAD53) +  (weightmatrixB .* fuD53);

clear weightmatrixA bandAD53 weightmatrixB bandmatch fuSAL3
clear bandASAL3


bandASAL4 = salience(bandAD54);
fuSAL4 = salience(fuD54);
bandmatch = match(bandAD54,fuD54,bandASAL4,fuSAL4);
[Y X] = size(bandmatch);
weightmatrixA = zeros(Y,X);
weightmatrixB = zeros(Y,X);
 alpha = .9
   for i = 1:Y
     for j = 1:X
        if bandmatch(i,j) < alpha
           if bandASAL4(i,j) > fuSAL4(i,j)
             weightmatrixA(i,j) = 1;
           else
             weightmatrixB(i,j) = 1;
           end
        else
```

```
          wmin = .5 - .5*((1-bandmatch(i,j))/(1-alpha));
          wmax = 1 - wmin;

          if bandASAL4(i,j) > fuSAL4(i,j)
            weightmatrixA(i,j) = wmax;
            weightmatrixB(i,j) = wmin;
          else
            weightmatrixA(i,j) = wmin;
            weightmatrixB(i,j) = wmax;
          end

   end
end
end

 fuD54 = (weightmatrixA .* bandAD54) +  (weightmatrixB .* fuD54);

clear weightmatrixA bandAD54 weightmatrixB bandmatch fuSAL4
clear bandASAL4

end %for loop

[OLD51 OLD52 OLD53 OLD54] = OLP(fuD51, fuD52, fuD53, fuD54,4);
clear fuD51 fuD52 fuD53 fuD54

[FSD5 ] = FSDLP(OLD51, OLD52, OLD53, OLD54);
clear OLD51 OLD52 OLD53 OLD54

[RELAP5 ] = RELP(FSD5,W);
clear FSD5

 % Now the reconstruction of the Image begins.  Using the results of
% equation (A3) Note: I use the original top level from the Gaussian
% Pyramid to start the reconstruction.  i.e. G6 = bandAG6
band30G0 = getim_sar('km15hhl');
band30G1 = Grad_reduce(band30G0,W);
clear band30G0
band30G2 = Grad_reduce(band30G1,W);
clear band30G1
band30G3 = Grad_reduce(band30G2,W);
clear band30G2
band30G4 = Grad_reduce(band30G3,W);
clear band30G3
band30G5 = Grad_reduce(band30G4,W);
clear band30G4
G6 = Grad_reduce(band30G5,W);
clear band30G5

G5 = RELAP5 + Gauss_expand(G6,W);
clear RELAP5 G6

G4 = RELAP4 + Gauss_expand(G5,W);
clear RELAP4 G5

G3 = RELAP3 + Gauss_expand(G4,W);
clear RELAP3 G4

G2 = RELAP2 + Gauss_expand(G3,W);
clear RELAP2 G3

G1 = RELAP1 + Gauss_expand(G2,W);
clear RELAP1 G2

G0 = clean_sar(RELAP + Gauss_expand(G1,W));
clear RELAP G1
```

```
save   fusarm15 G0

quit
```

E.4 Image Fusion Subroutines

E.4.1 Reduction Algorithm.

```
% This function takes in a layer from a gaussian pyramid and produces the
% output layer.  It takes the input layer and convolves it with the weight
% matrix in weights, which is a gaussian type gradient function, and then
% downsamples by a factor of 2.

function reduce = Grad_reduce(IMAGE,FIVECONVOLVEFUNCTION)
   temp1 = pad(IMAGE,FIVECONVOLVEFUNCTION);
   temp2 = conv2(temp1,FIVECONVOLVEFUNCTION,'valid');
    clear temp1
   [Y X] = size(temp2);
   reduce = temp2((1:2:Y),(1:2:X));
return
```

E.4.2 Oriented Gradient Pyramid Algorithm.

```
% This function takes in a layer from a gaussian pyramid and produces the
% orientation gradient pyramid.  It takes the input layer and convolves
% it with the kernels specified by d1-dn, where the letter 1 tells how many
% kernel will be used.  Each kernel represent a type of filter or wavelet.


function [D1, D2, D3, D4] = Orient_Grad_Pyramid(IMAGE,THREECONVOLVEFUNCTION,1)

    d1 = [1 -1];
    d2 = 1/(sqrt(2)) * [0 -1;
                        1  0];
    d3 = [-1;
           1];
    d4 = 1/(sqrt(2)) *[-1 0;
                       0 1];
 temp1 = pad(IMAGE,THREECONVOLVEFUNCTION);
 temp2 = conv2(temp1,THREECONVOLVEFUNCTION,'valid');
 clear temp1
 TEMP = temp2 + IMAGE;
 clear temp2

for i = 1:1

   if i == 1
      temp1 = pad(TEMP,d1);
      D1 = conv2(temp1,d1,'valid');

   elseif i == 2
      temp1 = pad(TEMP,d2);
      D2 = conv2(temp1,d2,'valid');

   elseif i == 3
      temp1 = pad(TEMP,d3);
      D3 = conv2(temp1,d3,'valid');

   else
```

```
      temp1 = pad(TEMP,d4);
      D4 = conv2(temp1,d4,'valid');

   end %end if
  end %end for

clear d1 d2 d3 d4 TEMP

return
```

E.4.3 Oriented Laplacian Pyramid Algorithm.

```
% This fuction takes in a layer from an oriented gradient pyramid and converts
% it to a second derivative pyramid (or oriented Laplacian Pyramid). It takes
% the input layer and convolves it with the kernels specified by d1-dn, where
% the letter 1 tells how many kernels will be used.  Each kernel represent a
% type of filter or wavelet. They may or may not be the valid ones used in
% creating the Oriented Gradient Pyramid.


function [D1, D2, D3, D4] = OLP(W, X, Y, Z ,1)

    d1 = [1 -1];
    d2 = 1/(sqrt(2)) * [0 -1;
                        1  0];
    d3 = [-1;
          1];
    d4 = 1/(sqrt(2)) *[-1 0;
                       0 1];

for i = 1:1

   if i == 1
      W = pad2(W,d1);
      D1 = conv2(W,((-1/8)*d1),'valid');

   elseif i == 2
      X = pad2(X,d2);
      D2 = conv2(X,((-1/8)*d2),'valid');

   elseif i == 3
      Y = pad2(Y,d3);
      D3 = conv2(Y,((-1/8)*d3),'valid');

   else
      Z = pad2(Z,d4);
      D4 = conv2(Z,((-1/8)*d4),'valid');

   end  %end if
  end  %end for

clear d1 d2 d3 d4

return
```

E.4.4 FSD Laplacian Pyramid Algorithm.

```
% This fuction takes in a layer from an oriented Laplacian pyramid and converts
% it to an FSD Laplacian Pyramid.  It takes the input layer and sums over all
% of the orientations.

function [FSD] = FSDLP(D1, D2, D3, D4)
```

```
      FSD = D1 + D2 + D3 + D4;

return
```

E.4.5 RE Laplacian Pyramid Algorithm.

```
% This fuction takes in a layer from an FSD Laplacian pyramid and converts
% it to an RE Laplacian Pyramid.  It takes the input layer and convolves it
% with the weight matrix (1 + W)  where W  is the original 5 x 5 Gaussian
% filter used to generate the Gaussian pyramid.


function [RELAP] = RELP(FSD,W)


    [A B] = size(W);
    C = zeros(A,B);
    C(3,3) = 1;
    CON1 = C + W;
    temp = pad(FSD,CON1);
    RELAP = conv2(temp,CON1,'valid');


 return
```

E.4.6 Salience Algorithm.

```
% INPUTS: single pyramid level
% OUTPUT: salience matrix

function SAL = salience(PYR1)

 P = [1];

   temp1 = PYR1.*PYR1;

   temp = pad(temp1,P);
   SAL = conv2(temp,P,'valid');

  return
```

E.4.7 Match Algorithm.

```
% INPUTS: single pyramid levels from 2 input images, and its salience matrix
% OUTPUT: matrix containing the match values


function MTCH = match(PYR1,PYR2,SPYR1,SPYR2)

 zeroprotection = .0000001;

    temp2 = PYR1.* PYR2;


   numerator = 2*temp2 +  zeroprotection;
   clear temp2

   denominator = SPYR1 + SPYR2 +  zeroprotection;

   MTCH = numerator./denominator;
```

```
return
```

E.4.8 Expand Algorithm.

```
% This function takes an input image (IMAGE) and an extrapolating kernel (W) and doubles the size
% of the input image.  The doubling is performed by first adding zeros to every other row and
% column and then convolving with the extrapolating kernel to fill in the zeros.  In most cases the
% kernel (W) was a Gaussian type of filter.

function EXP = Gauss_expand(IMAGE,W)

[Y,X] = size(IMAGE);

wider = zeros(Y,2*X);
wider(:,1:2:2*X) = IMAGE;
taller = zeros(2*Y,2*X);
taller(1:2:2*Y,:) = wider;
%fluffedandpadded = pad(taller,W);
%upsampled = conv2(fluffedandpadded,W,'valid');

W2 = 4*W;
temp = pad(taller,W2);
EXP = (conv2(temp,W2,'valid'));

return
```

E.4.9 Weight Matrrix Generation Code.

```
%  This is the weight matrix referred to in Burt's paper on image fusion.
%  it is from the appendix A, pg 182.  It will be used to generate the
%  Gaussian Gradient Pyramid.
%
  w = 1/16 * [1 2 1;
              2 4 2;
              1 2 1;];

%  W = w*w  (w convolved with w)
%

W = 1/256 * [ 1  4  6  4 1;
              4 16 24 16 4;
              6 24 36 24 6;
              4 16 24 16 4;
              1  4  6  4 1];


return
```

Appendix F. Contrast Sensitivity Matlab M-Files

F.1 Image Fusion Algorithm For Bands 30 60 and 90

```
% This program is used to tell the fuse program what set of images to fuse.  It calls the fuse program
% that is designed to fuse AVIRIS images.

bands = [431 432 433];

G0 = wfuse_ennormal(8,bands, 0.05);

save wband30noise G0

quit
```

F.2 Image Fusion Algorithm AVIRIS data

```
%INPUTS:
% step: The step size that the 40x40 window will be shifted.
% bands:  vector containing the AVIRIS images to be fused%
%OUTPUTS:  cleaned up version of the fused AVIRIS image.
%


% This algorithm performs a six layer decomposition fusion of the input data.
% The fusion weights are determined by  a weighting metric that is based upon the contrast
% sensitivity response of the analyst..  The weights are derived from the weights.m file, which uses
% the Peter Burt, wavelet filters and the  Contrast sensitivity numbers.  The file weightings.m
% computes the desired fusion weights for the details.  This program is specifically designed to
% fuse the AVIRIS images used in this thesis.  The images are also energy normalized before
% decomposition takes place.
%

function G0 = wfuse(step, bands, delta);
weights

[Y loop] = size(bands);

bandAG0 = ennormal(getim(bands(1)));

[fuD01 fuD02 fuD03 fuD04] = Orient_Grad_Pyramid(bandAG0,w,4);
bandAG1 = Grad_reduce(bandAG0,W);
clear bandAG0
[fuD11 fuD12 fuD13 fuD14] = Orient_Grad_Pyramid(bandAG1,w,4);
bandAG2 = Grad_reduce(bandAG1,W);
clear bandAG1
[fuD21 fuD22 fuD23 fuD24] = Orient_Grad_Pyramid(bandAG2,w,4);
bandAG3 = Grad_reduce(bandAG2,W);
clear bandAG2
[fuD31 fuD32 fuD33 fuD34] = Orient_Grad_Pyramid(bandAG3,w,4);
bandAG4 = Grad_reduce(bandAG3,W);
clear bandAG3
[fuD41 fuD42 fuD43 fuD44] = Orient_Grad_Pyramid(bandAG4,w,4);
bandAG5 = Grad_reduce(bandAG4,W);
clear bandAG4
[fuD51 fuD52 fuD53 fuD54] = Orient_Grad_Pyramid(bandAG5,w,4);
G6 = Grad_reduce(bandAG5,W);
clear bandAG5
```

```
for i = 2:loop,


bandBG0 = ennormal(getim(bands(i)));
band = bands(i)  % This is just for output messages to the logfile

level = 0

[bandBD01 bandBD02 bandBD03 bandBD04] = Orient_Grad_Pyramid(bandBG0,w,4);

w01  = detail_weightings(step,fuD01,bandBD01,C0, delta);
w0b = ones(size(w01)) - w01;
fuD01 = (w01 .* fuD01) +  (w0b .* bandBD01);
clear w01 w0b

w02  = detail_weightings(step,fuD02,bandBD02,C0, delta);
w0b = ones(size(w02)) - w02;
fuD02 = (w02 .* fuD02) +  (w0b .* bandBD02);
clear w02 w0b

w03  = detail_weightings(step,fuD03,bandBD03,C0, delta);
w0b = ones(size(w03)) - w03;
fuD03 = (w03 .* fuD03) +  (w0b .* bandBD03);
clear w03 w0b

w04  = detail_weightings(step,fuD04,bandBD04,C0, delta);
w0b = ones(size(w04)) - w04;
fuD04 = (w04 .* fuD04) +  (w0b .* bandBD04);
clear w03 w0b


level = 1

bandBG1 = Grad_reduce(bandBG0,W);
clear bandBG0

[bandBD11 bandBD12 bandBD13 bandBD14] = Orient_Grad_Pyramid(bandBG1,w,4);

w11  = detail_weightings(step,fuD11,bandBD11,C1, delta);
w1b = ones(size(w11)) - w11;
fuD11 = (w11 .* fuD11) +  (w1b .* bandBD11);
clear w11 w1b

w12  = detail_weightings(step,fuD12,bandBD12,C1, delta);
w1b = ones(size(w12)) - w12;
fuD12 = (w12 .* fuD12) +  (w1b .* bandBD12);
clear w12 w1b

w13  = detail_weightings(step,fuD13,bandBD13,C1, delta);
w1b = ones(size(w13)) - w13;
fuD13 = (w13 .* fuD13) +  (w1b .* bandBD13);
clear w13 w1b

w14  = detail_weightings(step,fuD14,bandBD14,C1, delta);
w1b = ones(size(w14)) - w14;
fuD14 = (w14 .* fuD14) +  (w1b .* bandBD14);
clear w13 w1b


level = 2

bandBG2 = Grad_reduce(bandBG1,W);
clear bandBG1

[bandBD21 bandBD22 bandBD23 bandBD24] = Orient_Grad_Pyramid(bandBG2,w,4);
```

```
w21  = detail_weightings(step,fuD21,bandBD21,C2, delta);
w2b = ones(size(w21)) - w21;
fuD21 = (w21 .* fuD21) +  (w2b .* bandBD21);
clear w21 w2b

w22  = detail_weightings(step,fuD22,bandBD22,C2, delta);
w2b = ones(size(w22)) - w22;
fuD22 = (w22 .* fuD22) +  (w2b .* bandBD22);
clear w22 w2b

w23  = detail_weightings(step,fuD23,bandBD23,C2, delta);
w2b = ones(size(w23)) - w23;
fuD23 = (w23 .* fuD23) +  (w2b .* bandBD23);
clear w23 w2b

w24  = detail_weightings(step,fuD24,bandBD24,C2, delta);
w2b = ones(size(w24)) - w24;
fuD24 = (w24 .* fuD24) +  (w2b .* bandBD24);
clear w23 w2b


level = 3

bandBG3 = Grad_reduce(bandBG2,W);
clear bandBG2

[bandBD31 bandBD32 bandBD33 bandBD34] = Orient_Grad_Pyramid(bandBG3,w,4);

w31  = detail_weightings(step,fuD31,bandBD31,C3, delta);
w3b = ones(size(w31)) - w31;
fuD31 = (w31 .* fuD31) +  (w3b .* bandBD31);
clear w31 w3b

w32  = detail_weightings(step,fuD32,bandBD32,C3, delta);
w3b = ones(size(w32)) - w32;
fuD32 = (w32 .* fuD32) +  (w3b .* bandBD32);
clear w32 w3b

w33  = detail_weightings(step,fuD33,bandBD33,C3, delta);
w3b = ones(size(w33)) - w33;
fuD33 = (w33 .* fuD33) +  (w3b .* bandBD33);
clear w33 w3b

w34  = detail_weightings(step,fuD34,bandBD34,C3, delta);
w3b = ones(size(w34)) - w34;
fuD34 = (w34 .* fuD34) +  (w3b .* bandBD34);
clear w33 w3b

level = 4

bandBG4 = Grad_reduce(bandBG3,W);
clear bandBG3

[bandBD41 bandBD42 bandBD43 bandBD44] = Orient_Grad_Pyramid(bandBG4,w,4);

w41  = detail_weightings(step,fuD41,bandBD41,C4, delta);
w4b = ones(size(w41)) - w41;
fuD41 = (w41 .* fuD41) +  (w4b .* bandBD41);
clear w41 w4b

w42  = detail_weightings(step,fuD42,bandBD42,C4, delta);
w4b = ones(size(w42)) - w42;
fuD42 = (w42 .* fuD42) +  (w4b .* bandBD42);
clear w42 w4b
```

```
w43  = detail_weightings(step,fuD43,bandBD43,C4, delta);
w4b = ones(size(w43)) - w43;
fuD43 = (w43 .* fuD43) +  (w4b .* bandBD43);
clear w43 w4b

w44  = detail_weightings(step,fuD44,bandBD44,C4, delta);
w4b = ones(size(w44)) - w44;
fuD44 = (w44 .* fuD44) +  (w4b .* bandBD44);
clear w43 w4b


level = 5

bandBG5 = Grad_reduce(bandBG4,W);
clear bandBG4

[bandBD51 bandBD52 bandBD53 bandBD54] = Orient_Grad_Pyramid(bandBG5,w,4);
clear bandBG5

w51  = detail_weightings(step,fuD51,bandBD51,C5, delta);
w5b = ones(size(w51)) - w51;
fuD51 = (w51 .* fuD51) +  (w5b .* bandBD51);
clear w51 w5b

w52  = detail_weightings(step,fuD52,bandBD52,C5, delta);
w5b = ones(size(w52)) - w52;
fuD52 = (w52 .* fuD52) +  (w5b .* bandBD52);
clear w52 w5b

w53  = detail_weightings(step,fuD53,bandBD53,C5, delta);
w5b = ones(size(w53)) - w53;
fuD53 = (w53 .* fuD53) +  (w5b .* bandBD53);
clear w53 w5b

w54  = detail_weightings(step,fuD54,bandBD54,C5, delta);
w5b = ones(size(w54)) - w54;
fuD54 = (w54 .* fuD54) +  (w5b .* bandBD54);
clear w53 w5b


% create the Orientation Gradient Pyramid with the 1st orientation filter d1.
% The levels are indexed by bandxij where i is the pyramid level and
% j is the orientation (d1,d2,...dn) (see fig 3 and equation A9).

% create the Orientation Laplacian Pyramid with the 1st orientation filter d1.
% The levels are indexed by bandxij where i is the pyramid level and
% j is the orientation (d1,d2,...dn) (see fig 3 and equation A9).

% create the FSD Laplacian Pyramid

% create the RE Laplacian Pyramid



end % end for i


% Now the reconstruction of the Image begins.  Using the results of
% equation (A3) Note: I use the original top level from the Gaussian
% Pyramid to start the reconstruction.  i.e. G4 = bandAG4

recreate = 1

[OLD01 OLD02 OLD03 OLD04] = OLP(fuD01, fuD02, fuD03, fuD04,4);
```

```
clear fuD01 fuD02 fuD03 fuD04

[FSD ] = FSDLP(OLD01, OLD02, OLD03, OLD04);
clear OLD01 OLD02 OLD03 OLD04

[RELAP ] = RELP(FSD,W);
clear FSD


[OLD11 OLD12 OLD13 OLD14] = OLP(fuD11, fuD12, fuD13, fuD14,4);
clear fuD11 fuD12 fuD13 fuD14

[FSD1] = FSDLP(OLD11, OLD12, OLD13, OLD14);
clear OLD11 OLD12 OLD13 OLD14

[RELAP1] = RELP(FSD1,W);
clear FSD1


[OLD21 OLD22 OLD23 OLD24] = OLP(fuD21, fuD22, fuD23, fuD24,4);
clear fuD21 fuD22 fuD23 fuD24

[FSD2] = FSDLP(OLD21, OLD22, OLD23, OLD24);
clear OLD21 OLD22 OLD23 OLD24

[RELAP2] = RELP(FSD2,W);
clear FSD2


[OLD31 OLD32 OLD33 OLD34] = OLP(fuD31, fuD32, fuD33, fuD34,4);
clear fuD31 fuD32 fuD33 fuD34

[FSD3] = FSDLP(OLD31, OLD32, OLD33, OLD34);
clear OLD31 OLD32 OLD33 OLD34

[RELAP3] = RELP(FSD3,W);
clear FSD3


[OLD41 OLD42 OLD43 OLD44] = OLP(fuD41, fuD42, fuD43, fuD44,4);
clear fuD41 fuD42 fuD43 fuD44

[FSD4 ] = FSDLP(OLD41, OLD42, OLD43, OLD44);
clear OLD41 OLD42 OLD43 OLD44

[RELAP4 ] = RELP(FSD4,W);
clear FSD4


[OLD51 OLD52 OLD53 OLD54] = OLP(fuD51, fuD52, fuD53, fuD54,4);
clear fuD51 fuD52 fuD53 fuD54

[FSD5 ] = FSDLP(OLD51, OLD52, OLD53, OLD54);
clear OLD51 OLD52 OLD53 OLD54

[RELAP5 ] = RELP(FSD5,W);
clear FSD5


G5 = RELAP5 + Gauss_expand(G6,W);
clear RELAP5 G6

G4 = RELAP4 + Gauss_expand(G5,W);
clear RELAP4 G5
```

```
G3 = RELAP3 + Gauss_expand(G4,W);
clear RELAP3 G4

G2 = RELAP2 + Gauss_expand(G3,W);
clear RELAP2 G3

G1 = RELAP1 + Gauss_expand(G2,W);
clear RELAP1 G2

G0 = RELAP + Gauss_expand(G1,W);
clear RELAP G1

G0 = clean(G0);

return
```

F.3 Image Fusion of Lenna Test Images

```
% This program is used to tell the fuse program what set of images to fuse.  It calls the
%fuse program that is designed to fuse Lenna images.

bands = [331 332 333];
G0 = wfuse512(8,bands, 0.05);
save new_lennanoise G0
quit
```

F.4 Image Fusion Algorithm For Lenna Test Images

```
%INPUTS:
% step: The step size that the 40x40 window will be shifted.
% bands:  vector containing the image files to be fused
%OUTPUTS:  cleaned up version of the fused  images.
%
% This algorithm performs a six layer decomposition fusion of the input data.
% The fusion weights are determined by  a weighting metric that is based upon the contrast
% sensitivity response of the analyst..  The weights are derived from the weights.m file, which uses
% the Peter Burt, wavelet filters and the  Contrast sensitivity numbers.  The file weightings.m
% computes the desired fusion weights for the details.  This program is specifically designed to
% fuse the Lenna   images used in this thesis.
%
function G0 = wfuse(step, bands, delta);
weights

[Y loop] = size(bands);

imagea = bands(1);
eval(['bandAG0 = getim512(' ' imagea ' ');']);

[fuD01 fuD02 fuD03 fuD04] = Orient_Grad_Pyramid(bandAG0,w,4);
bandAG1 = Grad_reduce(bandAG0,W);
clear bandAG0
[fuD11 fuD12 fuD13 fuD14] = Orient_Grad_Pyramid(bandAG1,w,4);
bandAG2 = Grad_reduce(bandAG1,W);
clear bandAG1
[fuD21 fuD22 fuD23 fuD24] = Orient_Grad_Pyramid(bandAG2,w,4);
bandAG3 = Grad_reduce(bandAG2,W);
clear bandAG2
[fuD31 fuD32 fuD33 fuD34] = Orient_Grad_Pyramid(bandAG3,w,4);
bandAG4 = Grad_reduce(bandAG3,W);
clear bandAG3
[fuD41 fuD42 fuD43 fuD44] = Orient_Grad_Pyramid(bandAG4,w,4);
```

```
bandAG5 = Grad_reduce(bandAG4,W);
clear bandAG4
[fuD51 fuD52 fuD53 fuD54] = Orient_Grad_Pyramid(bandAG5,w,4);
G6 = Grad_reduce(bandAG5,W);
clear bandAG5


for i = 2:loop,

imageb = bands(i);
eval(['bandBG0 = getim512(' ' imageb ' ');']);
band = bands(i)  % This is just for output messages to the logfile

level = 0

[bandBD01 bandBD02 bandBD03 bandBD04] = Orient_Grad_Pyramid(bandBG0,w,4);

w01  = detail_weightings(step,fuD01,bandBD01,C0,delta);
w0b = ones(size(w01)) - w01;
fuD01 = (w01 .* fuD01) +  (w0b .* bandBD01);
clear w01 w0b

w02  = detail_weightings(step,fuD02,bandBD02,C0,delta);
w0b = ones(size(w02)) - w02;
fuD02 = (w02 .* fuD02) +  (w0b .* bandBD02);
clear w02 w0b

w03  = detail_weightings(step,fuD03,bandBD03,C0,delta);
w0b = ones(size(w03)) - w03;
fuD03 = (w03 .* fuD03) +  (w0b .* bandBD03);
clear w03 w0b

w04  = detail_weightings(step,fuD04,bandBD04,C0,delta);
w0b = ones(size(w04)) - w04;
fuD04 = (w04 .* fuD04) +  (w0b .* bandBD04);
clear w03 w0b


level = 1

bandBG1 = Grad_reduce(bandBG0,W);
clear bandBG0

[bandBD11 bandBD12 bandBD13 bandBD14] = Orient_Grad_Pyramid(bandBG1,w,4);

w11  = detail_weightings(step,fuD11,bandBD11,C1,delta);
w1b = ones(size(w11)) - w11;
fuD11 = (w11 .* fuD11) +  (w1b .* bandBD11);
clear w11 w1b

w12  = detail_weightings(step,fuD12,bandBD12,C1,delta);
w1b = ones(size(w12)) - w12;
fuD12 = (w12 .* fuD12) +  (w1b .* bandBD12);
clear w12 w1b

w13  = detail_weightings(step,fuD13,bandBD13,C1,delta);
w1b = ones(size(w13)) - w13;
fuD13 = (w13 .* fuD13) +  (w1b .* bandBD13);
clear w13 w1b

w14  = detail_weightings(step,fuD14,bandBD14,C1,delta);
w1b = ones(size(w14)) - w14;
fuD14 = (w14 .* fuD14) +  (w1b .* bandBD14);
clear w13 w1b
```

```
level = 2

bandBG2 = Grad_reduce(bandBG1,W);
clear bandBG1

[bandBD21 bandBD22 bandBD23 bandBD24] = Orient_Grad_Pyramid(bandBG2,w,4);

w21  = detail_weightings(step,fuD21,bandBD21,C2,delta);
w2b = ones(size(w21)) - w21;
fuD21 = (w21 .* fuD21) +  (w2b .* bandBD21);
clear w21 w2b

w22  = detail_weightings(step,fuD22,bandBD22,C2,delta);
w2b = ones(size(w22)) - w22;
fuD22 = (w22 .* fuD22) +  (w2b .* bandBD22);
clear w22 w2b

w23  = detail_weightings(step,fuD23,bandBD23,C2,delta);
w2b = ones(size(w23)) - w23;
fuD23 = (w23 .* fuD23) +  (w2b .* bandBD23);
clear w23 w2b

w24  = detail_weightings(step,fuD24,bandBD24,C2,delta);
w2b = ones(size(w24)) - w24;
fuD24 = (w24 .* fuD24) +  (w2b .* bandBD24);
clear w23 w2b


level = 3

bandBG3 = Grad_reduce(bandBG2,W);
clear bandBG2

[bandBD31 bandBD32 bandBD33 bandBD34] = Orient_Grad_Pyramid(bandBG3,w,4);

w31  = detail_weightings(step,fuD31,bandBD31,C3,delta);
w3b = ones(size(w31)) - w31;
fuD31 = (w31 .* fuD31) +  (w3b .* bandBD31);
clear w31 w3b

w32  = detail_weightings(step,fuD32,bandBD32,C3,delta);
w3b = ones(size(w32)) - w32;
fuD32 = (w32 .* fuD32) +  (w3b .* bandBD32);
clear w32 w3b

w33  = detail_weightings(step,fuD33,bandBD33,C3,delta);
w3b = ones(size(w33)) - w33;
fuD33 = (w33 .* fuD33) +  (w3b .* bandBD33);
clear w33 w3b

w34  = detail_weightings(step,fuD34,bandBD34,C3,delta);
w3b = ones(size(w34)) - w34;
fuD34 = (w34 .* fuD34) +  (w3b .* bandBD34);
clear w33 w3b

level = 4

bandBG4 = Grad_reduce(bandBG3,W);
clear bandBG3

[bandBD41 bandBD42 bandBD43 bandBD44] = Orient_Grad_Pyramid(bandBG4,w,4);

w41  = detail_weightings(step,fuD41,bandBD41,C4,delta);
w4b = ones(size(w41)) - w41;
```

```
fuD41 = (w41 .* fuD41) +  (w4b .* bandBD41);
clear w41 w4b

w42  = detail_weightings(step,fuD42,bandBD42,C4,delta);
w4b = ones(size(w42)) - w42;
fuD42 = (w42 .* fuD42) +  (w4b .* bandBD42);
clear w42 w4b

w43  = detail_weightings(step,fuD43,bandBD43,C4,delta);
w4b = ones(size(w43)) - w43;
fuD43 = (w43 .* fuD43) +  (w4b .* bandBD43);
clear w43 w4b

w44  = detail_weightings(step,fuD44,bandBD44,C4,delta);
w4b = ones(size(w44)) - w44;
fuD44 = (w44 .* fuD44) +  (w4b .* bandBD44);
clear w43 w4b


level = 5

bandBG5 = Grad_reduce(bandBG4,W);
clear bandBG4

[bandBD51 bandBD52 bandBD53 bandBD54] = Orient_Grad_Pyramid(bandBG5,w,4);
clear bandBG5

w51  = detail_weightings(step,fuD51,bandBD51,C5,delta);
w5b = ones(size(w51)) - w51;
fuD51 = (w51 .* fuD51) +  (w5b .* bandBD51);
clear w51 w5b

w52  = detail_weightings(step,fuD52,bandBD52,C5,delta);
w5b = ones(size(w52)) - w52;
fuD52 = (w52 .* fuD52) +  (w5b .* bandBD52);
clear w52 w5b

w53  = detail_weightings(step,fuD53,bandBD53,C5,delta);
w5b = ones(size(w53)) - w53;
fuD53 = (w53 .* fuD53) +  (w5b .* bandBD53);
clear w53 w5b

w54  = detail_weightings(step,fuD54,bandBD54,C5,delta);
w5b = ones(size(w54)) - w54;
fuD54 = (w54 .* fuD54) +  (w5b .* bandBD54);
clear w53 w5b


% create the Orientation Gradient Pyramid with the 1st orientation filter d1.
% The levels are indexed by bandxij where i is the pyramid level and
% j is the orientation (d1,d2,...dn) (see fig 3 and equation A9).

% create the Orientation Laplacian Pyramid with the 1st orientation filter d1.
% The levels are indexed by bandxij where i is the pyramid level and
% j is the orientation (d1,d2,...dn) (see fig 3 and equation A9).

% create the FSD Laplacian Pyramid

% create the RE Laplacian Pyramid



end % end for i
```

```
% Now the reconstruction of the Image begins.  Using the results of
% equation (A3) Note: I use the original top level from the Gaussian
% Pyramid to start the reconstruction.  i.e. G4 = bandAG4

recreate = 1

[OLD01 OLD02 OLD03 OLD04] = OLP(fuD01, fuD02, fuD03, fuD04,4);
clear fuD01 fuD02 fuD03 fuD04

[FSD ] = FSDLP(OLD01, OLD02, OLD03, OLD04);
clear OLD01 OLD02 OLD03 OLD04

[RELAP ] = RELP(FSD,W);
clear FSD


[OLD11 OLD12 OLD13 OLD14] = OLP(fuD11, fuD12, fuD13, fuD14,4);
clear fuD11 fuD12 fuD13 fuD14

[FSD1] = FSDLP(OLD11, OLD12, OLD13, OLD14);
clear OLD11 OLD12 OLD13 OLD14

[RELAP1] = RELP(FSD1,W);
clear FSD1


[OLD21 OLD22 OLD23 OLD24] = OLP(fuD21, fuD22, fuD23, fuD24,4);
clear fuD21 fuD22 fuD23 fuD24

[FSD2] = FSDLP(OLD21, OLD22, OLD23, OLD24);
clear OLD21 OLD22 OLD23 OLD24

[RELAP2] = RELP(FSD2,W);
clear FSD2


[OLD31 OLD32 OLD33 OLD34] = OLP(fuD31, fuD32, fuD33, fuD34,4);
clear fuD31 fuD32 fuD33 fuD34

[FSD3] = FSDLP(OLD31, OLD32, OLD33, OLD34);
clear OLD31 OLD32 OLD33 OLD34

[RELAP3] = RELP(FSD3,W);
clear FSD3


[OLD41 OLD42 OLD43 OLD44] = OLP(fuD41, fuD42, fuD43, fuD44,4);
clear fuD41 fuD42 fuD43 fuD44

[FSD4 ] = FSDLP(OLD41, OLD42, OLD43, OLD44);
clear OLD41 OLD42 OLD43 OLD44

[RELAP4 ] = RELP(FSD4,W);
clear FSD4


[OLD51 OLD52 OLD53 OLD54] = OLP(fuD51, fuD52, fuD53, fuD54,4);
clear fuD51 fuD52 fuD53 fuD54

[FSD5 ] = FSDLP(OLD51, OLD52, OLD53, OLD54);
clear OLD51 OLD52 OLD53 OLD54

[RELAP5 ] = RELP(FSD5,W);
clear FSD5
```

```
G5 = RELAP5 + Gauss_expand(G6,W);
clear RELAP5 G6

G4 = RELAP4 + Gauss_expand(G5,W);
clear RELAP4 G5

G3 = RELAP3 + Gauss_expand(G4,W);
clear RELAP3 G4

G2 = RELAP2 + Gauss_expand(G3,W);
clear RELAP2 G3

G1 = RELAP1 + Gauss_expand(G2,W);
clear RELAP1 G2

G0 = RELAP + Gauss_expand(G1,W);
clear RELAP G1

G0 = clean512(G0);

return
```

F.5 Image Fusion of SAR Test images

```
% This program is used to tell the fuse program what set of images to fuse.  It calls the fuse program
% that is designed to fuse SAR images.

bands = ['km15hhl'; 'km15hvl'; 'km15vvl'];

G0 = wfuse_sar(8,bands, 0.05);

save fused_sarm15 G0

quit
```

F.6 Image Fusion Algorithm For SAR Imagery

```
%INPUTS:
% step: The step size that the 40x40 window will be shifted.
% bands:  vector containing the sarfiles to be fused%
%OUTPUTS:  cleaned up version of the fused sar image.
%


% This algorithm performs a six layer decomposition fusion of the input data.
% The fusion weights are determined by  a weighting metric that is based upon the contrast
% sensitivity response of the analyst..  The weights are derived from the weights.m file, which uses
% the Peter Burt, wavelet filters and the  Contrast sensitivity numbers.  The file weightings.m
% computes the desired fusion weights for the details.  This program is specifically designed to
% fuse the SAR images used in this thesis.   %

function G0 = wfuse_sar(step, bands, delta);
weights

[loop X] = size(bands);

imagea = bands(1,:);
eval(['bandAG0 = getim_sar(' ' imagea ' ');']);

[fuD01 fuD02 fuD03 fuD04] = Orient_Grad_Pyramid(bandAG0,w,4);
```

```
bandAG1 = Grad_reduce(bandAG0,W);
clear bandAG0
[fuD11 fuD12 fuD13 fuD14] = Orient_Grad_Pyramid(bandAG1,w,4);
bandAG2 = Grad_reduce(bandAG1,W);
clear bandAG1
[fuD21 fuD22 fuD23 fuD24] = Orient_Grad_Pyramid(bandAG2,w,4);
bandAG3 = Grad_reduce(bandAG2,W);
clear bandAG2
[fuD31 fuD32 fuD33 fuD34] = Orient_Grad_Pyramid(bandAG3,w,4);
bandAG4 = Grad_reduce(bandAG3,W);
clear bandAG3
[fuD41 fuD42 fuD43 fuD44] = Orient_Grad_Pyramid(bandAG4,w,4);
bandAG5 = Grad_reduce(bandAG4,W);
clear bandAG4
[fuD51 fuD52 fuD53 fuD54] = Orient_Grad_Pyramid(bandAG5,w,4);
G6 = Grad_reduce(bandAG5,W);
clear bandAG5


for i = 2:loop,

imageb = bands(i,:);
eval(['bandBG0 = getim_sar(' ' imageb ' ');']);
band = bands(i,:)  % This is just for output messages to the logfile

level = 0

[bandBD01 bandBD02 bandBD03 bandBD04] = Orient_Grad_Pyramid(bandBG0,w,4);

w01  = detail_weightings(step,fuD01,bandBD01,C0, delta);
w0b = ones(size(w01)) - w01;
fuD01 = (w01 .* fuD01) +  (w0b .* bandBD01);
clear w01 w0b

w02  = detail_weightings(step,fuD02,bandBD02,C0,delta);
w0b = ones(size(w02)) - w02;
fuD02 = (w02 .* fuD02) +  (w0b .* bandBD02);
clear w02 w0b

w03  = detail_weightings(step,fuD03,bandBD03,C0,delta);
w0b = ones(size(w03)) - w03;
fuD03 = (w03 .* fuD03) +  (w0b .* bandBD03);
clear w03 w0b

w04  = detail_weightings(step,fuD04,bandBD04,C0,delta);
w0b = ones(size(w04)) - w04;
fuD04 = (w04 .* fuD04) +  (w0b .* bandBD04);
clear w03 w0b


level = 1

bandBG1 = Grad_reduce(bandBG0,W);
clear bandBG0

[bandBD11 bandBD12 bandBD13 bandBD14] = Orient_Grad_Pyramid(bandBG1,w,4);

w11  = detail_weightings(step,fuD11,bandBD11,C1,delta);
w1b = ones(size(w11)) - w11;
fuD11 = (w11 .* fuD11) +  (w1b .* bandBD11);
clear w11 w1b

w12  = detail_weightings(step,fuD12,bandBD12,C1,delta);
w1b = ones(size(w12)) - w12;
fuD12 = (w12 .* fuD12) +  (w1b .* bandBD12);
```

```
clear w12 w1b

w13  = detail_weightings(step,fuD13,bandBD13,C1,delta);
w1b = ones(size(w13)) - w13;
fuD13 = (w13 .* fuD13) +  (w1b .* bandBD13);
clear w13 w1b

w14  = detail_weightings(step,fuD14,bandBD14,C1,delta);
w1b = ones(size(w14)) - w14;
fuD14 = (w14 .* fuD14) +  (w1b .* bandBD14);
clear w13 w1b


level = 2

bandBG2 = Grad_reduce(bandBG1,W);
clear bandBG1

[bandBD21 bandBD22 bandBD23 bandBD24] = Orient_Grad_Pyramid(bandBG2,w,4);

w21  = detail_weightings(step,fuD21,bandBD21,C2,delta);
w2b = ones(size(w21)) - w21;
fuD21 = (w21 .* fuD21) +  (w2b .* bandBD21);
clear w21 w2b

w22  = detail_weightings(step,fuD22,bandBD22,C2,delta);
w2b = ones(size(w22)) - w22;
fuD22 = (w22 .* fuD22) +  (w2b .* bandBD22);
clear w22 w2b

w23  = detail_weightings(step,fuD23,bandBD23,C2,delta);
w2b = ones(size(w23)) - w23;
fuD23 = (w23 .* fuD23) +  (w2b .* bandBD23);
clear w23 w2b

w24  = detail_weightings(step,fuD24,bandBD24,C2,delta);
w2b = ones(size(w24)) - w24;
fuD24 = (w24 .* fuD24) +  (w2b .* bandBD24);
clear w23 w2b


level = 3

bandBG3 = Grad_reduce(bandBG2,W);
clear bandBG2

[bandBD31 bandBD32 bandBD33 bandBD34] = Orient_Grad_Pyramid(bandBG3,w,4);

w31  = detail_weightings(step,fuD31,bandBD31,C3,delta);
w3b = ones(size(w31)) - w31;
fuD31 = (w31 .* fuD31) +  (w3b .* bandBD31);
clear w31 w3b

w32  = detail_weightings(step,fuD32,bandBD32,C3,delta);
w3b = ones(size(w32)) - w32;
fuD32 = (w32 .* fuD32) +  (w3b .* bandBD32);
clear w32 w3b

w33  = detail_weightings(step,fuD33,bandBD33,C3,delta);
w3b = ones(size(w33)) - w33;
fuD33 = (w33 .* fuD33) +  (w3b .* bandBD33);
clear w33 w3b

w34  = detail_weightings(step,fuD34,bandBD34,C3,delta);
w3b = ones(size(w34)) - w34;
```

```
fuD34 = (w34 .* fuD34) +  (w3b .* bandBD34);
clear w33 w3b

level = 4

bandBG4 = Grad_reduce(bandBG3,W);
clear bandBG3

[bandBD41 bandBD42 bandBD43 bandBD44] = Orient_Grad_Pyramid(bandBG4,w,4);

w41  = detail_weightings(step,fuD41,bandBD41,C4,delta);
w4b = ones(size(w41)) - w41;
fuD41 = (w41 .* fuD41) +  (w4b .* bandBD41);
clear w41 w4b

w42  = detail_weightings(step,fuD42,bandBD42,C4,delta);
w4b = ones(size(w42)) - w42;
fuD42 = (w42 .* fuD42) +  (w4b .* bandBD42);
clear w42 w4b

w43  = detail_weightings(step,fuD43,bandBD43,C4,delta);
w4b = ones(size(w43)) - w43;
fuD43 = (w43 .* fuD43) +  (w4b .* bandBD43);
clear w43 w4b

w44  = detail_weightings(step,fuD44,bandBD44,C4,delta);
w4b = ones(size(w44)) - w44;
fuD44 = (w44 .* fuD44) +  (w4b .* bandBD44);
clear w43 w4b


level = 5

bandBG5 = Grad_reduce(bandBG4,W);
clear bandBG4

[bandBD51 bandBD52 bandBD53 bandBD54] = Orient_Grad_Pyramid(bandBG5,w,4);
clear bandBG5

w51  = detail_weightings(step,fuD51,bandBD51,C5,delta);
w5b = ones(size(w51)) - w51;
fuD51 = (w51 .* fuD51) +  (w5b .* bandBD51);
clear w51 w5b

w52  = detail_weightings(step,fuD52,bandBD52,C5,delta);
w5b = ones(size(w52)) - w52;
fuD52 = (w52 .* fuD52) +  (w5b .* bandBD52);
clear w52 w5b

w53  = detail_weightings(step,fuD53,bandBD53,C5,delta);
w5b = ones(size(w53)) - w53;
fuD53 = (w53 .* fuD53) +  (w5b .* bandBD53);
clear w53 w5b

w54  = detail_weightings(step,fuD54,bandBD54,C5,delta);
w5b = ones(size(w54)) - w54;
fuD54 = (w54 .* fuD54) +  (w5b .* bandBD54);
clear w53 w5b


% create the Orientation Gradient Pyramid with the 1st orientation filter d1.
% The levels are indexed by bandxij where i is the pyramid level and
% j is the orientation (d1,d2,...dn) (see fig 3 and equation A9).

% create the Orientation Laplacian Pyramid with the 1st orientation filter d1.
```

```
% The levels are indexed by bandxij where i is the pyramid level and
% j is the orientation (d1,d2,...dn) (see fig 3 and equation A9).

% create the FSD Laplacian Pyramid

% create the RE Laplacian Pyramid



end % end for i


% Now the reconstruction of the Image begins.  Using the results of
% equation (A3) Note: I use the original top level from the Gaussian
% Pyramid to start the reconstruction.  i.e. G4 = bandAG4

recreate = 1

[OLD01 OLD02 OLD03 OLD04] = OLP(fuD01, fuD02, fuD03, fuD04,4);
clear fuD01 fuD02 fuD03 fuD04

[FSD ] = FSDLP(OLD01, OLD02, OLD03, OLD04);
clear OLD01 OLD02 OLD03 OLD04

[RELAP ] = RELP(FSD,W);
clear FSD


[OLD11 OLD12 OLD13 OLD14] = OLP(fuD11, fuD12, fuD13, fuD14,4);
clear fuD11 fuD12 fuD13 fuD14

[FSD1] = FSDLP(OLD11, OLD12, OLD13, OLD14);
clear OLD11 OLD12 OLD13 OLD14

[RELAP1] = RELP(FSD1,W);
clear FSD1


[OLD21 OLD22 OLD23 OLD24] = OLP(fuD21, fuD22, fuD23, fuD24,4);
clear fuD21 fuD22 fuD23 fuD24

[FSD2] = FSDLP(OLD21, OLD22, OLD23, OLD24);
clear OLD21 OLD22 OLD23 OLD24

[RELAP2] = RELP(FSD2,W);
clear FSD2


[OLD31 OLD32 OLD33 OLD34] = OLP(fuD31, fuD32, fuD33, fuD34,4);
clear fuD31 fuD32 fuD33 fuD34

[FSD3] = FSDLP(OLD31, OLD32, OLD33, OLD34);
clear OLD31 OLD32 OLD33 OLD34

[RELAP3] = RELP(FSD3,W);
clear FSD3


[OLD41 OLD42 OLD43 OLD44] = OLP(fuD41, fuD42, fuD43, fuD44,4);
clear fuD41 fuD42 fuD43 fuD44

[FSD4 ] = FSDLP(OLD41, OLD42, OLD43, OLD44);
clear OLD41 OLD42 OLD43 OLD44

[RELAP4 ] = RELP(FSD4,W);
```

```
clear FSD4


[OLD51 OLD52 OLD53 OLD54] = OLP(fuD51, fuD52, fuD53, fuD54,4);
clear fuD51 fuD52 fuD53 fuD54

[FSD5 ] = FSDLP(OLD51, OLD52, OLD53, OLD54);
clear OLD51 OLD52 OLD53 OLD54

[RELAP5 ] = RELP(FSD5,W);
clear FSD5

G5 = RELAP5 + Gauss_expand(G6,W);
clear RELAP5 G6

G4 = RELAP4 + Gauss_expand(G5,W);
clear RELAP4 G5

G3 = RELAP3 + Gauss_expand(G4,W);
clear RELAP3 G4

G2 = RELAP2 + Gauss_expand(G3,W);
clear RELAP2 G3

G1 = RELAP1 + Gauss_expand(G2,W);
clear RELAP1 G2

G0 = RELAP + Gauss_expand(G1,W);
clear RELAP G1

G0 = clean_sar(G0);
return
```

F.7 Image Fusion Subroutines

F.7.1 Reduction Algorithm.

```
% This fuction takes in a layer from a gaussian pyramid and produces the
% output layer.  It takes the input layer and convolves it with the weight
% matrix in weights, which is a gaussian type gradient fuction, and then
% downsamples by a factor of 2.

function reduce = Grad_reduce(IMAGE,FIVECONVOLVEFUNCTION)
   temp1 = pad(IMAGE,FIVECONVOLVEFUNCTION);
   temp2 = conv2(temp1,FIVECONVOLVEFUNCTION,'valid');
    clear temp1
   [Y X] = size(temp2);
   reduce = temp2((1:2:Y),(1:2:X));

 return
```

F.7.2 Oriented Gradient Pyramid Algorithm.

```
 % This fuction takes in a layer from a gaussian pyramid and produces the
% orientation gradient pyramid.  It takes the input layer and convolves
% it with the kernels specified by d1-dn, where the letter 1 tells how many
% kernel will be used.  Each kernel represent a type of filter or wavelet.
```

```
function [D1, D2, D3, D4] = Orient_Grad_Pyramid(IMAGE,THREECONVOLVEFUNCTION,1)

    d1 = [1 -1];
    d2 = 1/(sqrt(2)) * [0 -1;
                        1  0];
    d3 = [-1;
           1];
    d4 = 1/(sqrt(2)) *[-1 0;
                       0 1];
 temp1 = pad(IMAGE,THREECONVOLVEFUNCTION);
 temp2 = conv2(temp1,THREECONVOLVEFUNCTION,'valid');
 clear temp1
 TEMP = temp2 + IMAGE;
 clear temp2

for i = 1:1

   if i == 1
      temp1 = pad(TEMP,d1);
      D1 = conv2(temp1,d1,'valid');

   elseif i == 2
      temp1 = pad(TEMP,d2);
      D2 = conv2(temp1,d2,'valid');

   elseif i == 3
      temp1 = pad(TEMP,d3);
      D3 = conv2(temp1,d3,'valid');

   else
      temp1 = pad(TEMP,d4);
      D4 = conv2(temp1,d4,'valid');

   end %end if
  end %end for

clear d1 d2 d3 d4 TEMP

return
```

F.7.3 Oriented Laplacian Pyramid Algorithm.

```
% This fuction takes in a layer from an oriented gradient pyramid and converts
% it to a second derivative pyramid (or oriented Laplacian Pyramid). It takes
% the input layer and convolves it with the kernels specified by d1-dn, where
% the letter 1 tells how many kernels will be used.  Each kernel represent a
% type of filter or wavelet. They may or may not be the valid ones used in
% creating the Oriented Gradient Pyramid.


function [D1, D2, D3, D4] = OLP(W, X, Y, Z ,1)

    d1 = [1 -1];
    d2 = 1/(sqrt(2)) * [0 -1;
                        1  0];
    d3 = [-1;
           1];
    d4 = 1/(sqrt(2)) *[-1 0;
                       0 1];

for i = 1:1

   if i == 1
      W = pad2(W,d1);
```

```
      D1 = conv2(W,((-1/8)*d1),'valid');

   elseif i == 2
      X = pad2(X,d2);
      D2 = conv2(X,((-1/8)*d2),'valid');

   elseif i == 3
      Y = pad2(Y,d3);
      D3 = conv2(Y,((-1/8)*d3),'valid');

   else
      Z = pad2(Z,d4);
      D4 = conv2(Z,((-1/8)*d4),'valid');

   end  %end if
  end  %end for

clear d1 d2 d3 d4

return
```

F.7.4 FSD Laplacian Pyramid Algorithm.

```
% This fuction takes in a layer from an oriented Laplacian pyramid and converts
% it to an FSD Laplacian Pyramid.  It takes the input layer and sums over all
% of the orientations.

function [FSD] = FSDLP(D1, D2, D3, D4)

       FSD = D1 + D2 + D3 + D4;

return
```

F.7.5 RE Laplacian Pyramid Algorithm.

```
% This fuction takes in a layer from an FSD Laplacian pyramid and converts
% it to an RE Laplacian Pyramid.  It takes the input layer and convolves it
% with the weight matrix (1 + W)  where W  is the original 5 x 5 Gaussian
% filter used to generate the Gaussian pyramid.

function [RELAP] = RELP(FSD,W)
    [A B] = size(W);
    C = zeros(A,B);
    C(3,3) = 1;
    CON1 = C + W;
    temp = pad(FSD,CON1);
    RELAP = conv2(temp,CON1,'valid');
return
```

F.7.6 Expand Algorithm.

```
 % This function takes an input image (IMAGE) and an extrapolating kernel (W) and doubles the size
% of the input image.  The doubling is performed by first adding zeros to every other row and
% column and then convolving with the extrapolating kernel to fill in the zeros.  In most cases the
% kernel (W) was a Gaussian type of filter.

function EXP = Gauss_expand(IMAGE,W)

[Y,X] = size(IMAGE);
```

```
wider = zeros(Y,2*X);
wider(:,1:2:2*X) = IMAGE;
taller = zeros(2*Y,2*X);
taller(1:2:2*Y,:) = wider;
%fluffedandpadded = pad(taller,W);
%upsampled = conv2(fluffedandpadded,W,'valid');

W2 = 4*W;
temp = pad(taller,W2);
EXP = (conv2(temp,W2,'valid'));

return
```

F.7.7 Contrast Sensitivity Weight Matrix (C) Generation Code.

```
% This function creates the appropriately scaled C matrix to use at each level of image analysis.

tempC0 = zeros(64,64);
tempC0(1:21,1:21) = C;


tempC0ifft = ifft2(tempC0);
tempC1ifft = Grad_reduce(tempC0ifft,W);
tempC2ifft = Grad_reduce(tempC1ifft,W);
tempC3ifft = Grad_reduce(tempC2ifft,W);
tempC4ifft = Grad_reduce(tempC3ifft,W);
tempC5ifft = Grad_reduce(tempC4ifft,W);

tempC1 = abs(fft2(tempC1ifft));
tempC2 = abs(fft2(tempC2ifft));
tempC3 = abs(fft2(tempC3ifft));
tempC4 = abs(fft2(tempC4ifft));
tempC5 = abs(fft2(tempC5ifft));

C0 = tempC0(1:21,1:21);
C1 = tempC1(1:21,1:21);

C2 = zeros(21,21);
[Y X] = size(tempC2);
C2(1:Y,1:X) = tempC2;

C3 = zeros(21,21);
[Y X] = size(tempC3);
C3(1:Y,1:X) = tempC3;

C4 = zeros(21,21);
[Y X] = size(tempC4);
C4(1:Y,1:X) = tempC4;

C5 = zeros(21,21);
[Y X] = size(tempC5);
C5(1:Y,1:X) = tempC5;

clear C Y X
clear tempC0ifft tempC1ifft tempC2ifft tempC3ifft tempC4ifft tempC5ifft
clear tempC0 tempC1 tempC2 tempC3 tempC4 tempC5


if (0 > 1)

figure
mesh(C0)
figure
mesh(C1)
```

```
figure
mesh(C2)
figure
mesh(C3)
figure
mesh(C4)
figure
mesh(C5)


end  %end if
```

F.7.8 Weight Matrix Generation Code.

```
%  This is the weight matrix referred to in Burt's paper on image fusion.
%  it is from the appendix A, pg 182.  It will be used to generate the
%  Gaussian Gradient Pyramid.
%
  w = 1/16 * [1 2 1;
              2 4 2;
              1 2 1;];

%  W = w*w  (w convolved with w)
%

W = 1/256 * [ 1  4  6  4 1;
              4 16 24 16 4;
              6 24 36 24 6;
              4 16 24 16 4;
              1  4  6  4 1];

     d1 = [1 -1];
     d2 = 1/(sqrt(2)) * [0 -1;
                         1  0];
     d3 = [-1;
           1];
     d4 = 1/(sqrt(2)) *[-1 0;
                         0 1];
 X = [0 140 180 220 250 249 248 247 246 220 195 185 170 160 150 145 140 130 120 110 100];


C = zeros(21,21);

for m = 1:21
        for n = 1:21

        C(m,n) = sqrt((X(m)^2) + (X(n)^2));

        end %end for m

      end %end for n


wpad = zeros(64,64);
wpad(1:3,1:3) = w;

padd1 = zeros(64,64);
padd1(1,1:2) = d1;

padd2 = zeros(64,64);
padd2(1:2,1:2) = d2;

padd3 = zeros(64,64);
padd3(1:2,1) = d3;
```

```
padd4 = zeros(64,64);
padd4(1:2,1:2) = d4;


fftwpad = fft2(wpad);
fftpadd1 = fft2(padd1);
fftpadd2 = fft2(padd2);
fftpadd3 = fft2(padd3);
fftpadd4 = fft2(padd4);


magfftpadd1w = abs(fftpadd1 + fftpadd1.*fftwpad);
magfftpadd2w = abs(fftpadd2 + fftpadd2.*fftwpad);
magfftpadd3w = abs(fftpadd3 + fftpadd3.*fftwpad);
magfftpadd4w = abs(fftpadd4 + fftpadd4.*fftwpad);

D1 = magfftpadd1w(2:21,2:21);
D2 = magfftpadd2w(2:21,2:21);
D3 = magfftpadd3w(2:21,2:21);
D4 = magfftpadd4w(2:21,2:21);

createC


clear magfftpadd1w magfftpadd2w magfftpadd3w magfftpadd4w
clear fftwpad fftpadd1 fftpadd2 fftpadd3 fftpadd4
clear d1 d2 d3 d4 X ans padd1 padd2 padd3 padd4 wpad m n

return
```

F.7.9 Weighting Matrix Generation Code For the Detail Weightings.

```
% INPUTS:
% D1, D2: detail matrix from the oriented gradient pyramid 1 and 2
% C: contrast sensitivity matrix corresponding to the gradient level
% OUTPUT:
% w: weight matrix
%
%  This function analyzes a set of  details from two different orientation gradient pyramids to
% determine the weights that should be used to fuse them.


function w1  = detail_weightings(step,D1,D2,C, delta)

zeroprotection = .000000001;

 [Y X] = size(D1);

 D1norm = zeros(Y+40,X+40);
 D1norm(1:Y,1:X) = D1;
 clear  D1

 D2norm = zeros(Y+40,X+40);
 D2norm(1:Y,1:X) = D2;
 clear  D2

w1 = zeros(Y,X);

temp1 = zeros(64,64);
temp2 = zeros(64,64);

ycount = 0;
xcount = 0;
```

```
for i = 1:step:Y-step,
   ycount = ycount + 1

   for j = 1:step:X-step,
    xcount = xcount + 1;

    temp1(1:40,1:40) = D1norm(i:i+39,j:j+39);
    temp2(1:40,1:40) = D2norm(i:i+39,j:j+39);

    ffttemp1 = abs(fft2(temp1));
    ac1 = ffttemp1(1:21,1:21);
    ennormalac1 = ac1./(sqrt(sum(sum(ac1.^2))) + zeroprotection);

    ffttemp2 = abs(fft2(temp2));
    ac2 = ffttemp2(1:21,1:21);
    ennormalac2 = ac1./(sqrt(sum(sum(ac1.^2))) + zeroprotection);

    conweighteden1 = sum(sum(C.*ennormalac1)) + zeroprotection;
    conweighteden2 = sum(sum(C.*ennormalac2)) + zeroprotection;

    conweighteddiff = conweighteden1 - conweighteden2;



    numdiff = abs(conweighteddiff);

    denomsum = conweighteden1 + conweighteden2 + zeroprotection;

    percentdiff =  numdiff/denomsum;



    if (percentdiff > delta) & (conweighteddiff > 0)
      w1(i:i+(step-1),j:j+(step-1)) = ones(step,step);

    elseif (percentdiff < delta) & (conweighteddiff >= 0)
      w1(i:i+(step-1),j:j+(step-1)) = (1 - (conweighteden2/conweighteden1 * 0.5))*ones(step,step);

    elseif (percentdiff1 < delta) & (conweighteddiff < 0)
      w1(i:i+(step-1),j:j+(step-1)) = (1 - (conweighteden1/conweighteden2 * 0.5))*ones(step,step);

    end %if



    end %for j
     xcount = 0;

  end % for i

return
```

Vita

Captain Terry A. Wilson was born in Chicago Heights, Illinois on 05 January 1962. He enlisted in the Air Force in 1982 and was assigned to the Munitions Maintenance Squadron, Barksdale AFB, Lousianna. From 1982 to 1987 he worked as an Electronic Systems Analyst performing both Missile and Aircraft maintenance. He was accepted into the Airman's Education and Commissioning Program (AECP) in 1987 and was sent to the University of Florida for his undergraduate degree in Electrical Engineering. Upon Graduation in May of 1990, he received a commission from the Air Force's Officer Training School. Captain Wilson's first assignment, after being commissioned, was to the Ballistic Missile Organization, Norton AFB, California as an Acquisition and Engineering Project Manger. From 1990 to1993 Captain Wilson was the ICBM Code Processing System (ICPS) Project Officer for the Peacekeeper Rail Garrison (PKRG) and The Rapid Evaluation and Combat Targeting (REACT) Programs. In this capacity, Capt Wilson managed the cost and scheduling of several contracts, oversaw the design, production, and fielding of the ICPS portions of the PKRG and REACT programs, and developed the initial requirement specs for the ICPS changes in the Guidance Replacement Program (GRP).

Permanent address: Rt 1, Box 13R
Homer, LA 71040

Bibliography

1. "Playboy Magazine," November 1972.
2. Alexander Toet, L. J. van Ruyven, J. M. Valeton. "Merging Thermal and Visual Images by a Contrast Pyramid," *Optical Engineering*, *28*(7):789–792 (1989).
3. Burt, Peter J. and Edward H. Adelson. "The Laplacian Pyramid as a Compact Image Code," *IEEE Transactions on Communications*, *31*(4):532–540 (April 1983).
4. Burt, Peter J. and Edward H. Adelson. "Merging Images Through Pattern Pecomposition," *SPIE, Applications of digital image processing*, *575*(3):173–181 (1985).
5. Burt, Peter J. and Raymond J. Kolczynski. "Enhanced Image Capture Through Fusion," *1993 IEEE 4th International Conference on Computer Vision*, *4*:173–182 (1993).
6. Cowger, Ronald I. *A Measurement Of The Anisotropic Modulation Transfer Function Of The Extrafovial Human Visual System.* MS thesis, Air Force Institute of Technology, 1973.
7. Fielding, Kenneth Henry. *Spatio-Temporal Pattern Recognition Using Hidden Markov Models.* PhD dissertation, Air Force Institute of Technology, 1994.
8. Guidry, Roland D. *A High Resolution Measurement of Anisotropic Modulation Transfer Function Of The Human Visual System.* MS thesis, Air Force Institute of Technology, 1973.
9. III, Alexander Akerman. "Pyramidal Techniques for Multisensor Fusion," *Proceedings of SPIE - The International Society for Optical Engineering*, *1828*:124–131 (1993).
10. Jr., Mark W. Cannon. "A Study of Stimulus Range Effects in Free Modulus Magnitude Estimation of Contrast," *Vision Research*, *24*(9):1049–1055 (1984).
11. Mallat, Stephane G. "A Theory for Multiresolution Signal Decomposition: the Wavelet Representation," *IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 11* (July, 1989).
12. Peli, Eli. "Contrast in Complex Images," *Optical Society of America*, *7*(10):2032–2040 (October 1990).
13. Peli, Eli and others. "Contrast Sensitivity Functions for Analysis and Simulation of Visual Perception," *Optical Society of America*, *3*:126–129 (February 1990). Noninvasive Assessment of the Visual System 1990 Technical Digest Series.
14. Rinker, Jack N. *Hyperspectral Imagery - What Is It? - What Can It Do?.* DTIC AD-A231 164, U.S. Army Engineer Topographic Laboratories, 1990.
15. Rosenfeld, A., editor. *Multiresolution Image Processing and Analysis.* New York: Springer-Verlag, 1984.

16. Rubin, Gary S. and Ronald A. Schuchard. "Does Contrast Sensitivity Predict Face Recognition Performance in Low-Vision Observers," *Optical Society of America, 3*:130–133 (February 1990). Noninvasive Assessment of the Visual System 1990 Technical Digest Series.

17. Sanders, Mark S. and Ernest J. McCormick. *Human Factors in Engineering and Design* (7th Edition). New York: McGraw-Hill, Inc, 1993.

18. Toet, Alexander. "Hierarchical Image Fusion," *Machine Vision and Applications*, 1–11 (1990).

19. Toet, Alexander. "Multiscale Contrast Enhancement with Applicatiosn to Image Fusion," *Optical Engineering*, *31*(5):1026–1031 (May 1992).

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	December 1994	Master's Thesis

4. TITLE AND SUBTITLE

Perceptual Based Image Fusion
with Applications to Hyperspectral Image Data

5. FUNDING NUMBERS

6. AUTHOR(S)

Terry Allen Wilson

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology, WPAFB OH 45433-6583

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT/GE/ENG/94D-32

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Walt Wurst
Aeronautical Systems Center
ASC/REFF
Wright Patterson AFB, OH 45433-7106

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Distribution Unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Development of new imaging sensors has created a need for image processing techniques that can fuse images from different sensors or multiple images produced by the same sensor. The methods presented here focus on combining image data from the Airborne Visual and Infrared Imaging Spectrometer (AVIRIS) hyperspectral sensor into a single or smaller subset of images while maintaining the visual information necessary for human analysis. Three hierarchical multi-resolution image fusion techniques are implemented and tested using the AVIRIS image data and test images that contain various levels of correlated or uncorrelated noise. Two of the algorithms are published fusion methods that combine images from multiple sensors. The third method was developed to fuse any co-registered image data. This new method uses the spatial frequency response (contrast sensitivity) of the human visual system to determine which parts of the input images contain the salient features that need to be preserved in the composite image(s). After analyzing the signal-to-noise ratios and visual aesthetics of the fused images, contrast sensitivity based fusion is shown to provide excellent fusion results and, in every case, clearly outperformed the other two methods. Finally, as an illustrative example of how the fusion techniques are independent of the hyperspectral application, they are applied to fusing multiple polarimetric images from a Synthetic Aperture Radar to enhance automated targeting techniques.

14. SUBJECT TERMS

Multi-resolution analysis, Hyperspectral, Contrast sensitivity, AVIRIS, Image fusion, Wavelet analysis, Pyramid, Heirarchical

15. NUMBER OF PAGES

212

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL